LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

SÓLO PROGRAMADORES



Revista especializada para usuarios de PC

AÑO IV. Nº 29
1250 PTAS.
ARGENTINA 9'50 $
CHILE 3000 $
PORTUGAL 1500$

GRATIS CD-ROM CON:
LENGUAJE TCL/TK
COMPILADOR GNAT ADA
DEBIAN LINUX

BORLAND INTRABUILDER PROFESIONAL 1.0

COMUNICACIONES EN WINDOWS

CURSO DE ADA

PROGRAMACIÓN DE INTERFACES GRÁFICAS CON WINDOWS X (TCL/TK)

Y además:

Curso de Inteligencia Artificial
Grandes sistemas
Comparativa de distribuciones Linux

CURSOS PRÁCTICOS DE:
Tecnología Web
Visual Basic 4.0
Visual C ++
Programación básica



8 413042 303299 00029



Asistencia Técnica y Formación

Soporte Técnico Borland®

Estimado usuario.

El continuo avance en las tecnologías informáticas, nos ofrecen cada día unas herramientas de trabajo más sofisticadas, así como la necesidad de actualizar los conocimientos.

El Soporte Técnico Borland, está destinado para todos aquellos que trabajan con la informática de forma profesional. Nuestro labor es proporcionarles ayuda cualificada para obtener el mayor rendimiento de las herramientas, agilizar su trabajo evitándole perder su tiempo en encontrar soluciones, y procurarle el tránsito más rápido y fácil a éstas nuevas tecnologías.

Con este fin, queremos darle a conocer nuestros servicios, que abarcan asistencia técnica, y formación. Además, encontrará que inscribirse al Soporte Técnico, le supone otras muchas ventajas.

Para todos nuestros socios Cuenta E-mail y Kit de conexión
Internet Gratis

Si desea recibir información detallada, y Agenda de nuestros Seminarios, Cursos de programación, y presentaciones de productos, póngase en contacto con nosotros en:

http: //www.databasedm.es
(también en Infovía)
Tel.: 902 10 20 77
Fax: 902 10 20 66
E-mail: soporte@databasedm.es

Database DM





EDITORIAL

ENERO 1997. Número 29
SÓLO PROGRAMADORES es una publicación de Tower Communications

Director Editor
Antonio M. Ferrer Abelló
aferrer@towercom.es
Coordinador Técnico
Eduardo Toribio Pazos
Edición
Miguel Cabezuelo
Colaboradores
F. de la Villa, Fernando J. Echevarrieta, Pedro Antón. J. M. Martín, L. Martín, J. M. Peco, José C. Remiro, Jorge del Río, César Sánchez, R. Carballo, C. Sánchez, F. Bertran, M. Jesús Recio, Carlos Arias

Jefe de Diseño
Fernando García Santamaría
fegarcia@towercom.es
Maquetación
Clara Francés
Tratamiento de Imagen
María Arce Giménez

Publicidad
Erika de la Riva (Madrid)
Tel.: (91) 661 42 11
Publicidad
Papín Gallardo (Barcelona)
Tel.: (93) 213 42 29

Suscripciones
Isabel Bojo
Tel. (91) 661 42 11 Fax: (91) 661 43 86
suscrip@towercom.es

Filmación
Filma Dos S.L.
Impresión
G. Reunidas
Distribución
SGEL

Director General
Antonio M. Ferrer Abelló
Director Financiero
Francisco García Díaz de Liaño
Director de Producción
Carlos Peropadre
Directora Comercial
Carmina Ferrer
Director de Distribución
Enrique Cabezas García
Asesor Editorial
Mario de Luis

Redacción, Publicidad y Administración
C/ Aragoneses, 7
28100 Pol. Ind. Alcobendas (MADRID)
Telf.: (91) 661 42 11 / Fax: (91) 661 43 86

Depósito legal: M-26827-1994
ISSN: 1134-4792

EL CAMBIO

Escuchaba hace poco un anuncio de una conocida marca de automóviles donde anunciaban las nuevas versiones de uno de sus más afamados modelos. En el anuncio se animaba al potencial comprador a visitar los concesionarios de la marca, germana para más señas, para comprobar *in situ* los cambios en la nueva versión de su clásico modelo, para apostillar al final del comercial con la coletilla "vea que nada ha cambiado". Este anuncio refleja un poco la filosofía de los que formamos la redacción de la revista sobre Sólo Programadores. Empieza un nuevo año, y las publicaciones tienden a renovarse de arriba a abajo, pero nosotros creemos en otro tipo de cambio. También nosotros incorporamos nuevas secciones, por ejemplo TCL/TK, una herramienta muy a tener en cuenta, y la nueva sección dedicada a la Inteligencia Artificial. Además, en Febrero, posiblemente se incorpore alguna más. Pero no es la nueva Sólo Programadores. Es la misma que lleva ya 29 números con éste, apareciendo mes tras mes en los quioscos de toda España y parte de América del Sur.

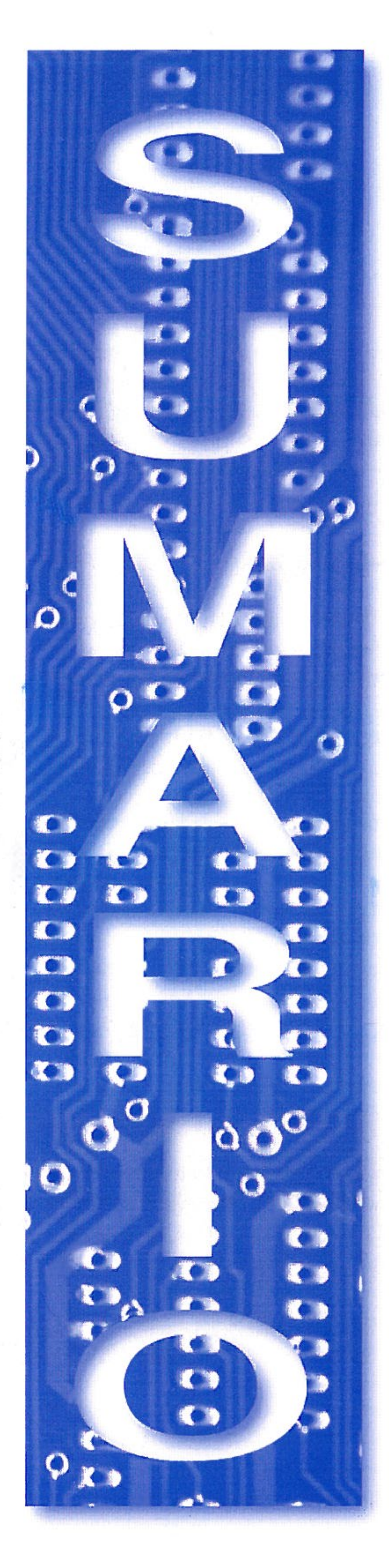
No, dicen que cuando algo funciona es mejor no tocarlo, y eso es lo que hacemos. Se incorporan gradualmente ciertas secciones que creemos tienen la suficiente demanda del lector, o la suficiente actualidad para entrar en nuestras páginas. Pero mantenemos la misma filosofía y estilo. En el 97 queremos seguir siendo una revista eminentemente técnica, orientada a la programación y con espacio a las nuevas tecnologías de la comunicación, orientada a profesionales y a los futuros profesionales.

Quiero ahora agradecer la continua recepción este pasado año de sugerencias y ofrecimientos para colaborar con nosotros de muchos lectores. Hoy por hoy, la mitad de nuestros colaboradores han surgido de esas cartas.

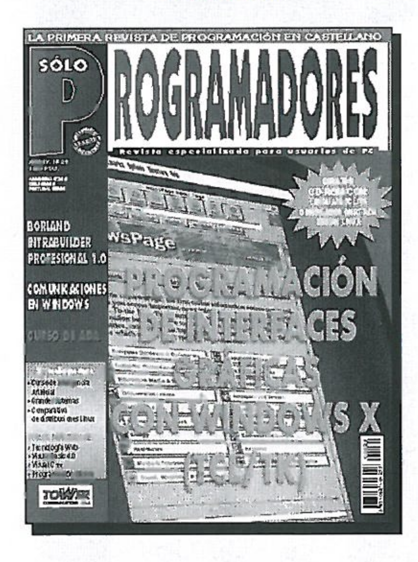
Volviendo a la revista , comentar que este mes incluimos en el CD-ROM la distribución Debian de Linux, un completo sistema operativo multitarea de verdad y completamente gratis, y la última versión de TCL/TK, con lo cual el lector podrá practicar en su PC lo mostrado en el curso de esta gran herramienta, tan potente para desarrollar aplicaciones como algunas de las más conocidas herramientas profesionales. Creo que no es fácil encontrar en el mercado de las revistas informáticas un CD de esta calidad.

En cuanto al contenido, además de lo comentado anteriormente, destacar que vuelve la sección de Linux tras un breve paréntesis, la segunda entrega de la sección dedicada a comunicaciones en Windows, un reportaje sobre Borland IntraBuilder, un análisis del compilador Tmt Pascal en la sección de Demos, ya que a partir de ahora se intentará que además de esta sección, otras utilicen en sus fuentes Pascal y este compilador, además del progresivo avance de las demás secciones. Feliz Año Nuevo a todos.

Eduardo Toribio
etori@ergos.es

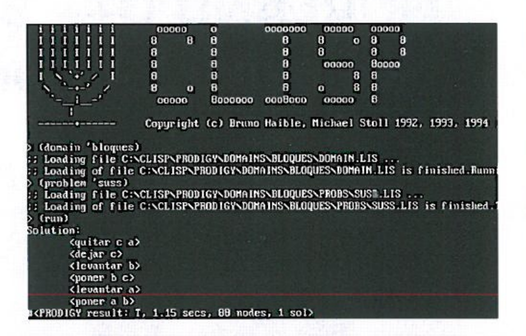

SUMARIO

TCL/TK



El tema de nuestra primera portada en este 97 es el nuevo curso dedicado a la programación en TCL. En este curso se tratarán los puntos más relevantes de este lenguaje, partiendo desde sus estructuras lógicas hasta la programación en red..

INTELIGENCIA ARTIFICIAL (Pág. X)



Con un nombre tan presuntuoso surgió la *Inteligencia Artificial* a la par que la Informática (antes, según las consideraciones), no se sabe si ciencia o ingeniería, y con el ambicioso objetivo de imitar la capacidad intelectual del hombre..

SÓLO PROGRAMADORES NOTICIAS

VISUAL BASIC 5.0 CONTROL. CREATION EDITION
Creación de controles ActiveX a través de Visual Basic

Microsoft ha anunciado recientemente Visual Basic 5.0 Control Creation Edition, una versión especializada de su conocido entorno visual de programación. Este nuevo paquete ha sido desarrollado para facilitar la creación de controles ActiveX de forma rápida y sencilla. De esta forma, los desarrolladores podrán fácilmente construir, probar, compilar y reutilizar controles ActiveX creados con esta herramienta e incluirlos en cualquier aplicación con soporte de esta tecnología, incluyendo Microsoft Internet Explorer, Office 97, Visual Basic 4.0, Visual C++, Visual FoxPro, Delphi y Netscape Navigator.



Visual Basic 5.0 Control Creation Edition incluye un nuevo editor de código, un creador de forms y un depurador interactivo, pero no incluirá en otras versiones de este paquete, como el Jet Database Engine.

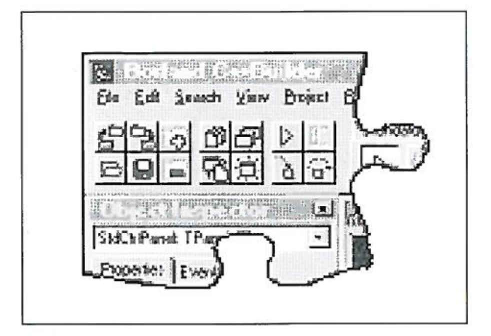
Visual Basic Control Creation Edition estará disponible por separado o junto con las versiones Standard, Professional y Enterprise de Visual Basic 5.0, de próxima aparición. Tanto la beta como las versiones finales de este producto están disponibles en la página Web de Microsoft en la dirección

http://www.microsoft.com/vbasic/
Para más información:

http://www.microsoft.com

BORLAND C++ BUILDER
Nueva línea de productos de desarrollo de Borland

Borland ha anunciado la próxima aparición de Borland C++ Builder, un nuevo producto de desarrollo no anunciado anteriormente. La compañía anunció también que el producto, conocido con el nombre clave Ebony, estará disponible durante el primer cuarto de 1997. Esta nueva herramienta proporcionará a los desarrolladores la potencia y el control de C++ combinada con la productividad de Delphi. Este paquete es un nuevo paso en la estrategia Golden Gate de Borland, destinada a rellenar el hueco existente entre arquitecturas cliente/servidor e Internet. Borland C++ Builder es una continuación de la familia de herramientas interoperables de Borland, que incluye Delphi Cliente/Servidor, Intrabuilder Cliente/Servidor y el próyimo entorno de desarrollo Java de Borland, OpenJBuilder. Como características principales del producto destacan la posibilidad de incluir componentes reutilizables, la velocidad que proporciona el desarrollo visual, la programación sencilla de bases de datos para aplicaciones cliente/servidor a través del Borland Database Engine (BDE), el soporte de los últimos estándares de la industria como el Win32 API, controles ActiveX y automatización OLE, conectividad de bases de datos ODBC y Winsock, así como el soporte de los API estándar de Internet Server y Netscape Server. Así mismo, el producto será totalmente complementario con Borland C++ y Delphi. El producto estará disponible también en versión cliente/servidor, y su precio aún no ha sido anunciado.



Para más información:

http://www.borland.com

C++ COMPONENTES SERIES VERSIÓN 2
Nuevas librerías de clases c++ de ObjetcSpace

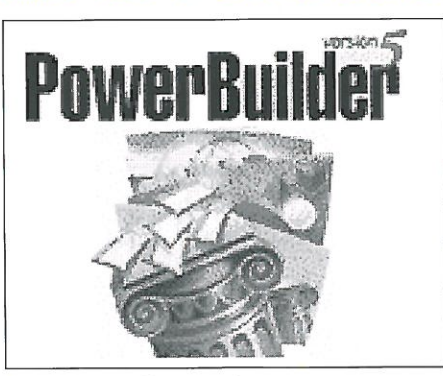
ObjectSpace Inc., fabricante de tecnología orientada a objetos, ha anunciado la segunda generación de su popular serie de componentes C++, incluyendo todas las nuevas versiones de Systems<ToolKit> y STL<ToolKit>. La versión 2 de C++ Components Series consiste en 10 librerías de clases C++, incluyendo la más portable versión de la librería ANSI/IO Standard C++ Library, disponible actualmente en el mercado. Esta segunda generación de librerías de clases de ObjectSpace ofrece a los programadores en C++ una espina dorsal estándar sobre la cual podrán desarrollar soluciones de negocios avanzadas. Esta serie de 10 librerías estará empaquetada en sus tres productos estrella: Systems<ToolKit>, STL<ToolKit> y Web<Toolkit>.

ObjectSpace C++ Components Series es totalmente portátil entre los compiladores, plataformas y sistemas operativos más conocidos. ObjectSpace proporciona la implementación más portátil de los estándares ANSI/IO disponibles actualmente en el mercado. Quizá el beneficio más obligatorio es la eficiencia y ejecución de las librerías de ObjectSpace. Todas estas librerías han sido integradas para su uso con los entornos de desarrollo integrados (IDE's) más comunes, como Visual C++ o Borland C++.

Para más información:
http://www.objectspace.com

INTERNET DEVELOPER TOOLKIT FOR POWERBUILDER 5.0
Nuevo Kit de herramientas de desarrollo de Powersoft

Powersoft, la división de herramientas de desarrollo de Sybase Inc., ha anunciado recientemente el lanzamiento de Internet Developer Toolkit for PowerBuilder 5.0. Como su propio nombre indica, se trata de un kit de herramientas de la compañía para PowerBuilder 5.0 para Windows. Este kit permite a los programadores en PowerBuilder crear y desplegar aplicaciones de bases de datos de negocios para Internet e intranets corporativas. Con Internet Developer Toolkit se podrá usar el entorno de desarrollo de PowerBuilder para crear servidores de aplicaciones personalizados que puedan generar dinámicamente páginas HTML para los navegadores más conocidos. Del mismo modo, también permite la creación de aplicaciones plug-in de Netscape para las intranet corporativas. Esta nueva herramienta responde a la demanda existente entre los programadores de herramientas que permitan extender las aplicaciones cliente/servidor a través de Internet, así como la creación de nuevas aplicaciones Web.



Para más información:
http://www.sybase.com
http://www.powersoft.com

CA-DOCVIEW/WEB
Visualizar documentos en Internet e intranet



Como continuación de su intento de unir las tecnologías de mainframes y plataformas cliente/servidor, Computer Associates Inc. ha anunciado CA-DocView/Web, solución de visualización de documentos para Internet e intranets. Al facilitar una manera sencilla de visualización de documentos, CA-DocView/Web ofrece una solución sencilla de usar para acceder a los datos de empresas alrededor de la red.

CA-DocView/Web explota completamente todas las capacidades del hypertexto ofrecidas en Internet. No sólo se pueden crear enlaces con informes desde cualquier documento Web, sino que los informes también pueden enlazar con cualquier documento Web, así como con otros informes. Además, incluye una serie de filtros que pueden crear automáticamente los links, eliminando la necesidad de programación adicional. Usando browsers estándar como Netscape Navigator o Microsoft Internet Explorer, CA-DocView/Web permite a los usuarios visualizar documentos completos o extraer secciones de documentos on-line. Así mismo, los informes largpos pueden ser leídos, analizados resumidos o contenidos fácilmente.

Ca-DocView soporta todas las direcciones de salida de CA y las soluciones de distribución de informes para conseguir la máxima flexibilidad. Al contrario que otros productos, que requieren un largo y tedioso proceso de carga de los datos, CA-DocView permite la rápida visualización de la salida de datos en un sólo paso.

Para más información:
http://www.cai.com

OBJECTIVE TOOLKIT 1.2
Librería de clases MFC de Stingray Software

Stingray Software ha anunciado la actualización de su extensión de librería de clases MFC Objective Toolkit 1.2. Esta actualización incrementa en cincuenta el número de clases del producto sin incremento de precio. Entre sus nuevas funcionalidades está la barra de status con contador de progreso y la justificación de texto; posibilidad de personalizar frames en una ventana; edición de cajas de diálogo y la posibilidad de añadir menús de herramientas en las aplicaciones, configurables por el usuario final y que permiten el uso de macros de Visual C++. Además, estas nuevas herramientas incluyen versiones beta de dos nuevas clases: Barras de herramientas personalizables y Control flexible de árboles. Objectiove Toolkit 1.2 incluye también varias mejoras para el kit, incluyendo el soporte de proyecciones 3D y mejoras en las clases Window. Objective Toolkit estará disponible a un precio aproximado de 60000 pesetas y soportará todas las versiones actuales de Microsoft Visual C++, incluyendo versiones de 16 y 32 bits.

Para más información:
http://www.stingsoft.com

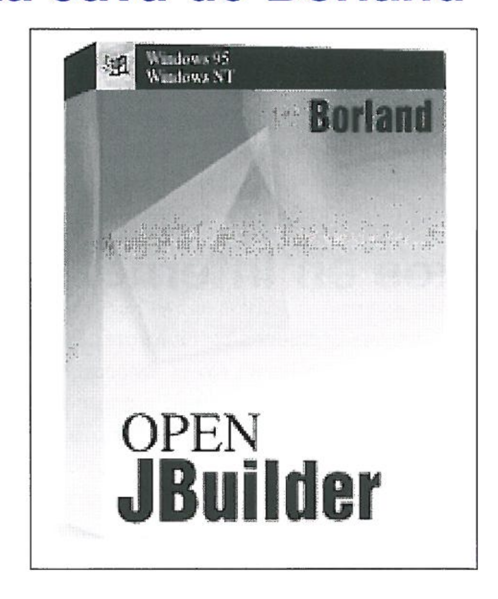
THE VICTOR IMAGE PROCESSING LIBRARY
Nueva librería de tratamiento de imágenes

Catenary Systems ha lanzado The Victor Image Processing Library, un conjunto de funciones para cerar aplicaciones de tratamiento de imágenes. Esta librería ofrece un potente proceso de imágenes, reducción de color, visualización, escaneo e impresión. Esta librería puede ser llamada desde cualquier compilador que soporte llamadas a librerías DLL en entornos de 16 y 32 bits y ofrece funciones para manipular casi todo tipo de formatos de imágenes binivel, en color o escala de grises, así como manipulación de brillo, contraste, filtros, escalado, rotado y operaciones matemáticas y lógicas con las imágenes. El paquete está disponible en versiones 16 bit (para Windows 3.1/3.11) y 32 bits (para entornos Windows 95/NT) e incluye la librería con un completo manual de usuario y referencias, así como una completa utilidad gráfica para tratar las imágenes y programas de ejemplo. Existe una demo disponible en la página Web de la compañía.

Para más información:
http://www.catenary.com

OPENJBUILDER
Nueva herramienta Java de Borland

Borland International ha anunciado recientemente la disponibilidad de una versión beta de su herramienta de programación Java OpenJBuilder. Con esta herramienta, Borland proporcionará a los programadores en Java una solución estándar de desarrollo para la construcción de aplicaciones Web multiplataforma. Además, su acuerdo con JavaSoft, Netscape y Microsoft asegura por mucho tiempo el soporte de las últimas tecnologías relacionadas con Java y ofrece a los desarrolladores la máxima flexibilidad y potencia en la creación y despliegue de sus aplicaciones. El software OpenJ Builder forma parte de su nueva generación de herramientas de desarrollo en Java de aplicaciones Web multiplataforma y ofrece un potente entorno de desarrollo para creación de applets, aplicaciones que requieran conectividad de bases de datos cliente/servidor, integración de hypertexto y soporte de componentes Java Beans reutilizables. Así mismo, OpenJ Builder incluye InterClient, un interesante conjunto de herramientas con un entorno abierto que proporcionará la posibilidad de incluir componentes y asistentes que ampliarán las capacidades de dichas herramientas. Al mismo tiempo, la compañia Borland ha anunciado también la dispinibilidad de su programa Open JBuilder Partner Program, diseñado para que los vendedores independientes de software desarrollen componentes JavaBeans. Para ello les proporcionarà versiones de evaluación de Open JBuilder, soporte técnico, documentación y otras ventajas. La versión beta de Open JBuilder se encuentra disponible desde hace tiempo en la página Web que tiene a disposiciòn del usuario la compañía.



Para más información:
http://www.borland.com

WEBDBC-ENTERPRISE 3.0
Desarrollo de bases de datos para Internet

Con el fin de crear un nuevo estándar en lo que a herramientas de desarrollo de bases de datos para Internet se refiere, StromCloud Development Corp. ha anunciado la próxima aparición de WebDBC-Enterprise 3.0 para Windows NT y Windows 95. Esta es una potente herramienta que facilitará a los programadores la creación de páginas HTML y el desarrollo de aplicaciones de bases de datos para Internet en Java y Visual Basic utilizando cualquier motor de bases de datos existente. Entre sus nuevas mejoras está el soporte de aplicaciones Visual Basic, componentes ActiveX (realizados pácticamente en cualquier lenguaje) y JDBC, que permite construir las páginas web más sofisticadas mezclando páginas web dinámicas y applets Java, así como soporte de automatización OLE y portabilidad multiplataforma (incluyendo entornos Unix y MacOS). Existen versiones disponibles de las herramientas de programación de la compañía en su página web.
Para más información:
http://www.stormcloud.com

OSMOS Y VISUALWAVE 2.0
Nuevo plan de distribución de Unisys

Unisys Corp. ha puesto en marcha recientemente un nuevo plan de distribución para OSMOS, su sistema de bases de datos relacionales orientado a objetos. Este plan consiste en la inclusión de una versión completamente funcional por un periodo de 90 días de OSMOS en el paquete VisualWave 2.0 de ParcPlace-Digitalk, permitiendo a sus compradores desarrollar plicaciones Web para Internet e intranets. VisualWave es un entorno integrado de desarrollo de aplicaciones dinámicas para la World Wide Web y servidores de intranets corporativas. De esta forma, la combinación de estos dos productos facilitará a los programadores desarrollar potentes aplicaciones escalables para la red. Así mosmo, la herramienta incluye el interfaz SmallTalk de ParcPlace-Digitalk, permitiendo que la integración con objetos de VisualWave se realice de la misma forma que con cualquier otro tipo de objetos.
Para más información:
http://www.osmos.com
http://www.parcplace.com

UNIXWARE 2.1.1
Nueva actualización del sistema operativo de SCO

Recientemente, SCO ha anunciado la disponibilidad de la nueva actualizacón de su sistema operativo corporativo, SCO UnixWare 2.1. Esta nueva versión, cuya revisión es 2.1.1, proporciona a los usuarios nuevos e importantes beneficios como el cumplimiento de la normativa UNIX 95, más amplio soporte de hardware, mayor rendimiento y modificaciones que abordan el problema del tratamiento de fechas para el año 2000.

Como es ya costumbre en la compañía, SCO continúa proporcionando a sus clientes las distintas actualizaciones de sus productos a través de Internet. De este modo, los usuarios actuales de UnixWare 2.1 pueden copiar la actualización, inmediatamente y sin coste, del servidor ftp de SCO (ftp://ftp.sco-com/UW21/upd 211) o pagar un derecho nominal por el coste de actualización. Los nuevos clientes del SCO UnixWare recibirán el CD de actualización con el producto original.

El paquete de actualización de UnixWare 2.1.1 contiene nuevas características para adaptarse a la normativa del UNIX95. La conformidad con este estándar proporciona una mayor flexibilidad y libertad de elección, asegurando la migración de las aplicaciones a través de una serie estándar de interfaces de programación de aplicaciones. Todo esto permitirá a los fabricantes de software incorporar al mercado más aplicaciones, de una forma más rápida y a un precio menor.

Asimismo, esta nueva revisión de SCO encara una de las preocupaciones del sector informático: el tratamiento de fechas para el año 2000. La nueva versión contiene comandos y utilidades actualizados que aseguran el correcto manejo de las fechas con más de dos dígitos y el proceso de cambio de año. Esta nueva característica ayudará a los usuarios a reducir los costes asociados con el problema del cambio del milenio y proporcionará mayor protección a la inversión realizada en las infraestructuras actuales.

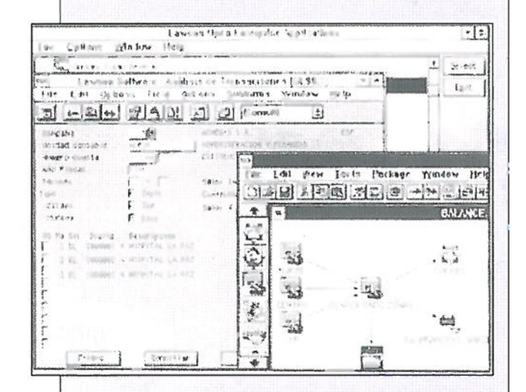
Además de esta serie de mejoras en el mantenimiento y rendimiento del sistema operativo, la actualización también incluye nuevo soporte de hardware. SCO ha ampliado el hardware soportado incluyendo los adaptadores SCSI y periféricos, nuevas tarjetas de vídeo y red, así como una nueva serie de mecanismos que permiten a los clientes aprovechar las últimas novedades en hardware.

Para más información:
http://www.sco.com/

NOVEDADES

LAWSON Software V 7.0

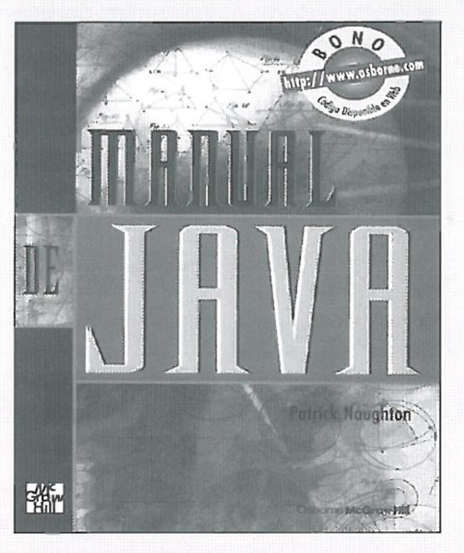
BURKE, distribuidor en España de Lawson Software, ha presentado la versión 7.0 de esta solución cliente/servidor para entornos de gestión y financieros. Este paquete permite utilizar bases de datos Microsoft SQL Server, además de las ya soportadas (Oracle, Sybase, Informix, etc...). Como novedad incorpora menús de proceso, entregando una librería de menús rediseñada o personalizada, pudiendo añadir aplicaciones no-Lawson a sus menús de todos los sistemas de negocio. Asimismo, incorpora capacidades de Workflow colaborativo. Las funciones de Workflow de la versión 7.0 proporcionarán tres modelos: Workflow personal, que integra aplicaciones en la estación



de trabajo, procesa pantallas y automatiza actividades; Workflow colaborativo, que combina y extiende el Workflow personal con correo electrónico y sistemas colaborativos como Lotus Notes; y Workflow de empresas, que proporciona control sobre los procesos globales de la compañía e integra los workflows anteriores.

LIBROS

Manual de JAVA



Java, el revolucionario entorno de programación creado por Sun Microsystems, está aportando nuevos niveles de interactividad a la World Wide Web. Con su poderosa animación, multimedia y características interactivas, Java ofrece un lenguaje de programación poderoso y flexible para Internet.

Esta obra está escrita por uno de los más famosos expertos mundiales en Java, fundador del equipo de desarrollo en Sun, y pretende transmitir todos los conocimientos posibles sobre qué puede hacer Java y cómo hacerlo.

En este manual de Java encontrará, además de todo lo necesario sobre cómo programar en Java, otro tipo de información como:

- Ejemplos de código para la creación de elementos interactivos en páginas Web.
- Lecciones de capacidades avanzadas de Java como multitarea, redes y programación GUI.
- Tratamiento completo de las bibliotecas del kit de desarrollo 1.0 de Java.

Editorial: Mc Graw-Hill
Autor: Patrick Naughton
395 páginas
Recomendado por Sólo Programadores
Idioma: Castellano
Nivel: Todos los niveles
Precio: 3.500 (I.V.A inc.)

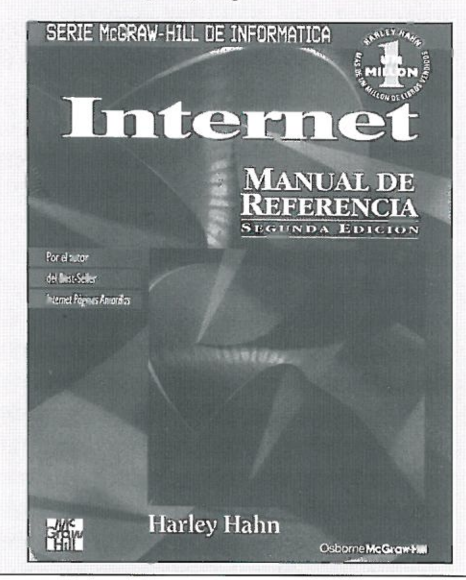
Internet manual de referencia

Que Internet es enorme y está creciendo es algo ya notorio para todos. Cómo estar al día es algo que se puede antojar difícil a aquellos que no tengan una relación directa con este medio. Para ello, Harley Hahn, uno de los autores más leídos en temas sobre Internet, ha desarrollado esta guía, segunda edición, completamente revisada y actualizada. En el libro encontrará :

- Qué hardware y software necesita para conectarse.
- Cómo usar Web, Gopher y FTP.
- Cómo comunicarse a través de correo electrónico.

El libro tiene un nivel adecuado para todo tipo de usuarios, y está orientado a todos aquellos que quieran dar sus primeros pasos en la red, o simplemente incrementar su nivel de conocimientos.

Editorial: Mc Graw-Hill
Autor: Harley Hahn
395 páginas
Idioma: Castellano
Nivel: Todos los niveles
Precio: Consultar editorial



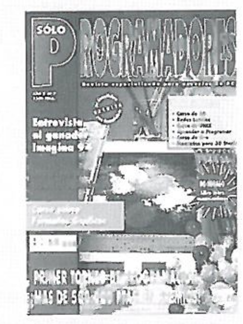
CÓMO SUSCRIBIRSE A



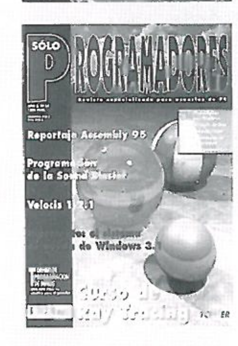


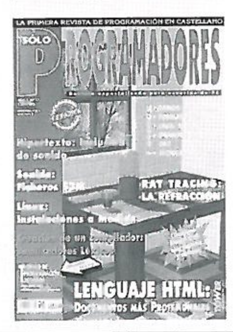












scríbase enviando este cupón por correo o fax (91) 661.43.86, o llamando al teléfono (91) 661.42.11 Horario 9 a 14 y 15:00 a 18:00 h.

eseo suscribirme a la revista SÓLO PROGRAMADORES acogiéndome a la siguiente modalidad:

☐ Suscripción: 1 año (12 números) **por sólo** 11.950 ptas. (ahorro 20%). ☐ Estudiantes carreras técnicas: 8.950 ptas. (ahorro 40%)

ESTA OFERTA ANULA LAS ANTERIORES, DESCUENTOS NO ACUMULABLES.

ombre y apellidos.. Domicilio..

oblación.. C.P .... Provincia........................ Telf.............................. Profesión...................

ORMA DE PAGO:

☐ Con cargo a mi tarjeta VISA nº ____ ____ ____ ____

Fecha de caducidad de la tarjeta................ Nombre del titular, si es distinto..

☐ Domiciliación bancaria.

Señor Director del banco ....

Población ....

Ruego a vd. que se sirva cargar en mi ☐ cuenta corriente ☐ libreta

ahorro número...

recibo que le será presentado por TOWER COMMUNICATIONS, S.R.L.

mo pago de mi suscripción a la revista SÓLO PROGRAMADORES.

CODIGO CUENTA CLIENTE

ENTIDAD	OFICINA	DC	Nº CUENTA

Firma:

☐ Contra-reembolso del importe más gastos de envío.

☐ Cheque a nombre de TOWER COMMUNICATIONS S.R.L., que adjunto.

☐ Giro Postal (adjunto fotocopia del resguardo).

Rellena este cupón y envíalo a:
TOWER COMMUNICATIONS SRL
C/ Aragoneses, 7
28100 Pol. Ind. ALCOBENDAS (Madrid)



CGI: PROGRAMACIÓN DE UN JUEGO INTERACTIVO

Fernando J. Echevarrieta

Tras algunos artículos tratando la interfaz CGI, se ha llegado a un punto en que el lector debe ser capaz de realizar cualquier aplicación WWW basada en esta interfaz. Este mes, como ejemplo, se aplicarán todos los conceptos expuestos para el desarrollo de un buscaminas a través del WWW.

En anteriores números se han expuesto la práctica totalidad de los aspectos que rodean la interfaz CGI, desde las propias especificaciones de la misma hasta la metodología que ésta impone. En esta entrega se pretende desarrollar una aplicación CGI compleja, que requiera hacer uso de todas esas técnicas y, lo que es más importante, que exija una labor de diseño y profundo análisis previa a su codificación. De esta forma, se irán presentando diversos problemas y se irá eligiendo entre las alternativas posibles, hasta llegar al desarrollo final de la aplicación.

Como ejemplo de aplicación WWW de cierta complejidad basada en CGI, se explicará el desarrollo de un juego interactivo. Para ello se partirá del juego del buscaminas, cuya programación como aplicación cliente-servidor se explicó en la sección de sistemas abiertos del número 18 de la revista [Echeva-1]. Para aquellos lectores que deseen probarlo, este juego se encuentra operativo en el servidor del autor.

Aquellos lectores que sigan las dos secciones obtendrán un aprovechamiento óptimo de estos ejemplos ya que, de esta forma, encajarán todas las piezas del puzzle y podrán terminar de asentar conceptos que pudieran haber quedado en el aire. En cualquier caso, no es en absoluto necesario haber siquiera leído aquel artículo para comprender el presente en su totalidad.

El ámbito en que se engloba el ejemplo es el siguiente:

Se trata de desarrollar una aplicación WWW consistente en un juego de tablero interactivo, en concreto, el juego del buscaminas. El problema a analizar en esta sección serán las restricciones de diseño del mismo, obviando la programación de los algoritmos y estudiando el problema desde un punto de vista genérico.

- El primer problema, y el más importante, es, como siempre, la conservación del estado de la partida. Una vez más se recuerda que HTTP no mantiene el estado entre una conexión y otra y, por tanto, será labor de la aplicación a desarrollar el mantener este estado e identificar cada una de las conexiones con la partida que les corresponde.

- El segundo problema, que con este planteamiento ya se deja ver, es que, como se mencionaba en [Echeva-2] y [Echeva-3], será necesario tener presente en todo momento que se debe realizar una aplicación que soporte el acceso concurrente, ya que hay que prever el caso en que se desarrollen diversas partidas de forma simultánea.

DIVISIÓN DE LA APLICACIÓN

Inicialmente se planteará el problema de la conservación del estado y posteriormente se ajustará para soportar el acceso concurrente. Así, siguiendo la metodología de diseño expuesta el mes pasado [Echeva-3], habrá que considerar una división de la aplicación considerando como fronteras aquellos puntos en los que se realizan operaciones de E/S. En el caso que nos ocupa, la entrada se reduce a aceptar la pulsación del ratón sobre la casilla que se

desea pisar, y la salida a dibujar el estado del tablero tras esta pulsación. A cada entrada siempre sucede una salida, por lo que éstos serán los puntos frontera de división en CGIs de la aplicación.

De esta forma, el mecanismo más lógico es considerar la aplicación como una sucesión de invocaciones a un CGI que recibe la información sobre la casilla que ha sido pisada, actualiza el estado del tablero y procede a generar el código HTML necesario para dibujar en la pantalla del cliente el tablero actualizado. Este código HTML deberá contener los enlaces necesarios para que, al pulsar el usuario sobre una casilla, se vuelva a invocar el CGI.

Por el momento, el planteamiento de la aplicación parece claro. Sin embargo, ahora que parece que hemos avanzado algo, nos encontramos con que no sólo no hemos resuelto los dos problemas iniciales, sino que ha surgido uno nuevo. En efecto, el mecanismo descrito muestra el estado de equilibrio estable del sistema, el transcurso de la partida. Pero ¿qué fue primero, el huevo o la gallina? El CGI toma como entrada la casilla pisada, pero al llamar al CGI por primera vez no hay dónde pisar porque no se ha llamado antes al CGI para que pinte el tablero que se debe pisar para llamar al CGI. Es decir, ¿quién dibujó el tablero cuyas casillas, al ser pisadas, invocan al CGI que dibuja el tablero? Habrá que contar con un método de inicialización del sistema.

EL ESTADO DE UNA PARTIDA

Como se explicaba, el CGI deberá actualizar el estado, pero ¿qué estado?. Al pintar el tablero, el CGI morirá, y con él su estado interno, por lo que habrá que preservarlo de algún modo. Como se analizaba el mes pasado, la preservación de estado entre procesos CGI se podía realizar mediante el empleo de diversas técnicas de transmisión directa o de almacenamiento externo. En el primero de los casos, se trataba de transmitir la información al CGI al invocarlo, por lo que se empleaban parámetros de línea de comando codificados en su URL de invocación o campos *HIDDEN* en el caso de *forms*. En el segundo, se empleaba el disco o un programa externo (demonio) para almacenar el estado antes de morir y permitir así que el siguiente proceso CGI lo recuperara.

En nuestro caso particular, no tiene sentido realizar una transmisión directa de estado entre CGIs para preservar el estado de la partida. En efecto, sería necesario, al menos, incluir en la línea de invocación al CGI un parámetro por cada casilla del tablero, por ejemplo:

<A HREF=/cgi-bin/minas?0+1+3+9+...>

donde 0 podría indicar que una casilla no tiene ni toca minas; 1, que toca una; 3 que toca 3...; 9 que tiene una mina... También habría que codificar si la casilla ha sido pisada o no, por ejemplo A, para la casilla que no tiene ni toca minas y ha sido pisada, etc.

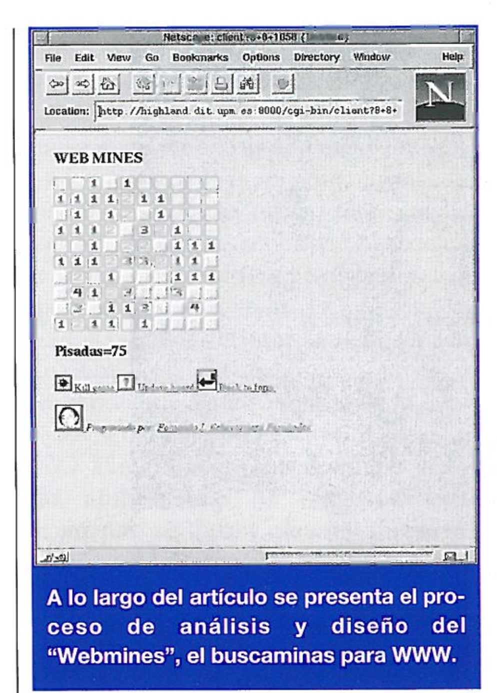
El método, que es válido, sólo presenta inconvenientes:

- La variable *QUERY_STRING* [Echeva-4] probablemente desbordaría.

- Cada enlace de invocación que apareciera en el código HTML generado para pintar el tablero debería contener la totalidad del estado del tablero. Dado que debe existir un enlace por cada casilla, el estado se encontraría repetido tantas veces como casillas hubiera, en lugar de almacenarse una sola vez. Así, para un tablero de 10x10 existirían 100 enlaces conteniendo cada uno de ellos la información del total del tablero. Es decir, unos 10Ks, sólo para transmitir la información del tablero. Información que debería ser transmitida del servidor al cliente cada vez que se actualizara el tablero, lo que ralentizaría enormemente la partida.

- A lo absurdo de las condiciones anteriores habría que añadir el hecho de que el usuario siempre podría consultar la fuente HTML para saber dónde están las minas.

Los principales factores en el desarrollo del "webmines" son la conservación de estado y la gestión de acceso concurrente



A lo largo del artículo se presenta el proceso de análisis y diseño del "Webmines", el buscaminas para WWW.

Por lo tanto, este mecanismo sólo resulta apropiado cuando la información de estado a transmitir ocupa unos pocos parámetros. En definitiva, habrá que recurrir a un método de almacenamiento externo.

LA IDENTIFICACIÓN DE CADA PARTIDA

Ante la alternativa de emplear un fichero en disco donde almacenar el estado del tablero o emplear un demonio para ello, la idea que se presenta más atractiva es la de emplear un fichero, que parece más sencillo. En efecto, se trata de la solución más limpia y sencilla siempre que uno se olvide de que está realizando programación concurrente. Uno no debe preocuparse si se le escapan estos detalles, el resultado será tan desastroso que pronto se dará cuenta. Así, si dos personas jugaran al buscaminas a la vez, ambas pisarían el mismo tablero con lo que, si no lo saben, obtendrían una visión de la par-

tida, al menos, desconcertante. Por tanto, será necesario proteger el acceso a los tableros.

Ya no basta con gestionar un acceso secuencial como en el caso de los contadores, sino que el estado deberá ser almacenado en zonas separadas. Como no es cuestión de programar un sistema gestor de bases de datos que trate con tableros, la idea sería emplear un fichero diferente para cada tablero. Perfecto, salvo que entonces habrá que nombrar de forma distinta a cada fichero y debe existir alguna forma de que cada CGI invocado sepa en qué partida se encuentra inmerso y cuál es el fichero que le corresponde.

Este problema existirá siempre, incluso cuando se trate con demonios, por lo que cada partida deberá identificarse de alguna forma. Esta identificación constituye asimismo información de estado que deberá salvaguardarse entre invocaciones a CGIs. Sin embargo, no se podrá almacenar junto a la del estado del tablero, ya que es precisamente el puntero que señala a esta última información. Por tanto, esta vez no queda otro remedio que incluir un identificador como parámetro en la URL correspondiente al enlace asociado a cada casilla del tablero, con lo que se realizará una transmisión de estado mediante parámetros. De esta forma se invocarán URLs de la forma siguiente:

/cgi-bin/client?<identificación de casilla>+<identificación_de_partida>

No debe confundirse lo que aquí se ha denominado *identificación de casilla* con el ejemplo que se ponía en el apartado anterior. En este caso se trata de identificar las coordenadas de la casilla pisada. Por ejemplo, en un tablero de 10x10 se podrían identificar las casillas numerándolas del 1 al 100 o mediante coordenadas desde (1,1) hasta (10,10). Éste último es el método por el que se ha optado en la aplicación ejemplo.

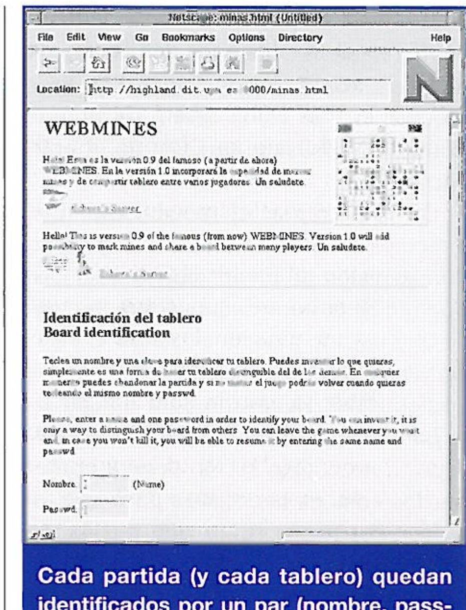
La conservación del estado de la aplicación se podrá realizar por transmisión entre CGIs o por almacenamiento externo

LA SOLUCIÓN ADOPTADA

Como se ha visto, la idea de emplear ficheros ya no resulta tan simple como parecía a primera vista, por lo que podría ser interesante estudiar el empleo de demonios externos. Por otra parte, en teoría, cada modificación o recuperación de estado requeriría un acceso al sistema de ficheros, lo que es dos órdenes de magnitud más lento que la comunicación con otro proceso. Además, en sistemas operativos como UNIX la forma de leer o escribir en un fichero es exactamente la misma que se emplea para leer o escribir en un *socket* que permitirá la comunicación con el demonio [Echeva-5] y [Echeva-1], con lo que el problema, en realidad, no es más complejo.

En el ejemplo que se facilita con la revista se ha optado por la solución del demonio, aunque se trata únicamente de eso, de un ejemplo. Lo que sí debe importar es el proceso de análisis y diseño que se lleva a cabo en cada aplicación WWW basada en CGI. En este tipo de aplicaciones es especialmente importante llevar a cabo estos pasos en primer lugar y proceder a la codificación únicamente cuando se tenga perfectamente definido el problema.

La solución más elegante habría sido realizar un único demonio que gestionara una lista de tableros. Cuando se recibiera una llamada de un CGI que no correspondiera a una partida en curso (es decir, a uno de los tableros existentes en la lista), se generaría un nuevo nodo en esta lista y se le asignaría un identificador que se devolvería a éste CGI. A partir de este momento, los sucesivos CGIs se transmitirían esta información mediante parámetros.



Cada partida (y cada tablero) quedan identificados por un par (nombre, password) que introduce el usuario mediante un form.

Sin embargo, se ha optado por una solución bastante menos "elegante" con dos finalidades:

- Por una parte, con el objetivo de modificar en lo mínimo posible el servidor que se desarrolló en [Echeva-1], norma que habrá que adoptar cuando se tenga que adaptar un programa ya existente para que pueda ser accedido desde un cliente WWW.
- Por otra parte, con el objetivo de emplear también el disco como almacen de estado, con lo que, en esta aplicación, se emplearían ya todas las técnicas presentadas en la serie (salvo campos ocultos que, en este caso, no proceden).

Así pues, en lugar de disponer de un servidor de puerto conocido, con el que todos los CGIs se ponen en contacto quedando diferenciadas sus partidas por un número identificador que se propaga entre invocaciones, se dispondrá de un servidor por cada partida diferente, siendo el puerto de atención de este servidor el elemento diferenciador entre partidas.

Como se indicaba anteriormente, será necesario dibujar un primer tablero cuyas casillas sean enlaces de invocación al primer CGI. El problema



surge de que estos enlaces deberán incluir ya el identificador del puerto de atención del servidor correspondiente a la partida. Por tanto, será necesario conocerlo de alguna forma. Si se empleara la solución "elegante" se podría colocar cualquier valor en el primer tablero. Siendo el puerto del servidor conocido, al recibir el CGI el identificador cuando se conectara a este puerto, lo incluiría en los enlaces del siguiente tablero que generara.

Sin embargo, con la solución adoptada ni siquiera se conoce el puerto donde preguntar. Por tanto, se vuelve a la idea de emplear un fichero para almacenar esta información. Así, se puede pedir al usuario mediante un *form* (figura 1) que identifique sus partidas con un nombre y un *password*, y disponer de un fichero en disco que relacione cada pareja de nombre y *password* con un puerto de atención. El fichero puede ser único y conocido por todos, ya que se trata simplemente de una tabla índice.

Esta aproximación permite a la vez que un usuario abandone una partida sin concluir y la continúe días despues. La aplicación no notará la diferencia, ya que cada "pisada" implica el nacimiento y muerte de una conexión HTTP y un script CGI y no importa si transcurre un segundo o un año entre dos conexiones, siempre que el demonio que guarda el tablero siga vivo. Lo que era un problema, se ha convertido en una ventaja. Así, días después de haber interrumpido una partida será posible continuarla introduciendo la misma pareja de nombre y *password* en el *form*. Al recibir el CGI el estado de la partida lo recibirá tal y como se dejó la última vez.

Claro, que para que el CGI reciba el estado, debe enviar las coordenadas de una casilla pisada. Por eso, se ha contemplado la posibilidad de enviar coordenadas imposibles, como (0,0) (si se numera desde (1,1)) para solicitar el estado del tablero sin realizar ninguna pisada. Otra coordenada imposible que se emplea es la (1,0), que provoca la muerte inmediata del servidor.

FUNCIONAMIENTO DE WEBMINES

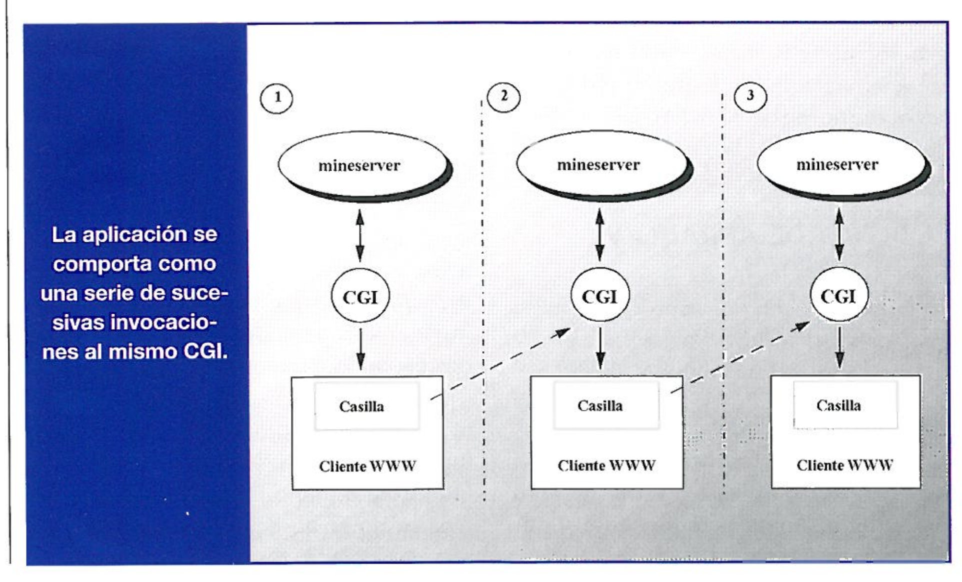
Al realizar la división de la aplicación, se definía ésta como una sucesión de invocaciones a un CGI. El CGI recibirá las coordenadas de la casilla pisada y actualizará el estado del tablero, procediendo posteriormente a mostrarlo en pantalla del cliente de WWW. El estado será almacenado por un demonio externo al que se ha denominado *mineserver* (figura 2). De este modo, el CGI al que se ha denominado *client* enviará al servidor las coordenadas de la casilla pisada y esperará a que éste le responda con la descripción de todo el tablero. Así pues, la labor del CGI es transmitir unas coordenadas y generar el código HTML necesario para dibujar un tablero, pero en ningún momento depende su funcionamiento de las coordenadas recibidas ni modifica por sí mismo estado alguno. Por ello, todos los algoritmos de juego se encuentran en el servidor, tratándose el CGI de una simple pasarela y, por ello, el conjunto CGI-servidor no es una verdadera aplicación cliente-servidor. La justificación de esto, así como las reglas que rigen esta comunicación y su programación, se describen exhaustivamente en [Echeva-1].

De esta forma, la ejecución de una partida se llevará a cabo tal y como se observa en la figuras 3 y 4. En la primera de ellas se han representado tres instantes correspondientes a 3 movimientos. Como se puede observar, la pulsación por parte del usuario de una casilla implica una nueva invocación del CGI (flecha discontinua) al que se le pasan como parámetros las coordenadas de la casilla pisada y el puerto de atención del servidor asociado a esa partida. Por ello, cada casilla debe ser un enlace HTML.

Lo lógico será que el juego se desarrolle de forma gráfica. Por tanto, existirán dos formas de abordar la construc-

La aplicación queda dividida en un CGI, client, encargado de recuperar el estado del tablero y dibujarlo en pantalla y un servidor, mineserver, encargado de actualizar y conservar el tablero.

HOST SERVIDOR DE WWW
WEBMINES
mineserver
Demonio
client
CGI
HOST DEL USUARIO
Cliente WWW



La aplicación se comporta como una serie de sucesivas invocaciones al mismo CGI.

ción del tablero: como una única imagen-mapa con una zona caliente asociada a cada casilla o como una multitud de imágenes enlace, una para cada casilla.

El primer planteamiento implica la necesidad de generar gráficos *on-line*, ya que no es posible disponer de antemano de las imágenes de todos los posibles tableros. Además de aumentar la complejidad del sistema, significa el envío de una imagen de superficie considerable en cada movimiento, imagen que nunca se encontrará en la caché del *browser* porque siempre será distinta.

Por el contrario, si se envía una imagen por cada casilla, aumentará el número de conexiones HTTP necesarias para dibujar los primeros tableros, aunque cada una de ellas transmitirá una imagen mucho menor. Cada una de estas imágenes corresponderá a una casilla sin pisar, vacía, indicando un uno, un dos... El primer tablero, además, sólo requerirá una conexión para recibir la imagen de casilla sin pisar, ya que se colocará la misma imagen repetidas veces. Conforme vayan llegando las imágenes, irán quedando en la caché, con lo que cada vez será necesario un número menor de conexiones, reduciéndose a cero cuando todas las imágenes estén en la caché.

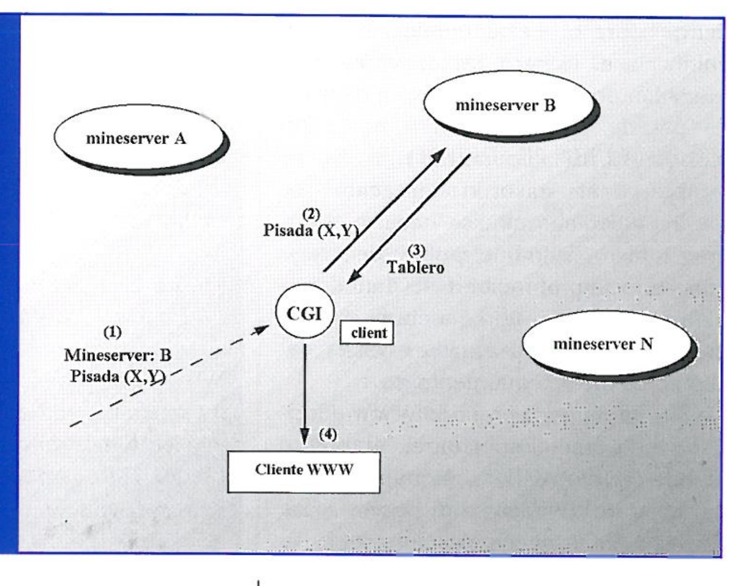
De esta forma, el CGI deberá generar por cada casilla un código HTML de la siguiente forma :

```
<A HREF=/cgi-bin/client?3+4+10037>
<IMG SRC=/minas/b.gif ALT=.>
</A>
```

que correspondería a la casilla (3,4), que presentaría una imagen de casilla sin pisar en un *browser* que mostrara imágenes y un punto en un *browser* en modo texto y que indicaría que el puerto de atención del servidor sería el 10037.

EL *SCRIPT* DE *ARRANQUE: CGISTARMINES*

Para dibujar el primer tablero e inicializar el sistema se emplea un *script* en *shell*. Si se ejecuta en local o en una red con baja latencia se comprobará la diferencia de velocidad entre los lenguajes *sh* y C a la hora de dibujar los tableros, ya que este primer tablero requiere un tiempo notablemente superior para aparecer.



El CGI es invocado recibiendo como parámetros las coordenadas de la casilla pisada, así como el puerto de atención del servidor del tablero (1), al que envía las coordenadas (2). Este le responde con el estado del tablero actualizado (3), tablero que el CGI dibujará en pantalla del cliente (4).

El *script* (figura 5) procesará la información que proviene de un *form* que emplea el método *POST*. Por tanto, leerá esta información de la entrada estándar (el código se encuentra en el disco de la revista). Dado que se trata únicamente de dos campos de texto y que se emplearán con labores de identificación, no es necesario recurrir a programas URL-decodificadores de ayuda. Mediante combinaciones de los comandos *echo*, *cut* y *sed* de UNIX se obtienen los valores deseados para el nombre y el *password* (1) que, de cara al usuario, identificarán la partida.

Una vez el *script* dispone de esta información, comprueba si existe la tabla índice que relaciona los nombres y *password* de cada tablero con el puerto de atención del demonio servidor que les corresponde y si ya existe una entrada para los facilitados. Si así ocurre, se tratará de una partida en curso, por lo que dará la oportunidad de reanudar el juego, destruir la partida o volver al *form* a teclear otra información. En los dos primeros casos ejecutara el procedimiento *yaexiste()* que leerá de la tabla el puerto por el que atiende el servidor (2) con objeto de incluirlo como parámetro en el URL de invocación asociado al enlace que corresponde a cada posibilidad (4).

Asimismo, como se puede observar, incluirá unas coordenadas de casillas imposibles (1,0) o (0,0) que el demonio interpretará como instrucciones de suicidio o de envío del estado sin realizar ninguna pisada, respectivamente.

```
<A HREF=/cgi-bin/client?1+0+$port>
<IMG SRC=/minas/mina.gif ALT=Kill game>
Kill game
</A><p>
```

En caso de que no exista ninguna partida asociada con el nombre y el *password* facilitados ejecutará el procedimiento *tablero()*, que se encarga de lanzar un demonio de tablero (3) al que facilita el nombre y el *password* junto con el nombre del fichero que actúa como tabla índice de tableros. El demonio, al arrancar, será el que actualice este fichero añadiendo una nueva entrada en la que incluirá el puerto de atención que le ha proporcionado el sistema operativo.

```
pid=getpid();
fd=fopen(file,"a+");
fprintf(fd,"%s,%s,%d,%d\n",nombre,password,ntohs(mineserver.sin_port),pid);
fclose(fd);
```

Asimismo, el propio procedimiento *tablero()* generará el código HTML correspondiente al primero de los tableros, con todas las casillas sin pisar, haciendo de cada una de ellas un enlace de invocación al CGI cliente que incluye las coordenadas de cada casilla, así como el puerto en que espera el servidor.

```
<A HREF=/cgi-bin/client?$x+$y+$port>
<img src=/minas/b.gif BORDER=0>
</A>
```

Para conocer este puerto, simplemente espera un segundo, dejando



tiempo para que el demonio inserte el valor en el fichero tabla, y lee del mismo.

EL CGI CLIENTE: CLIENT

El CGI cliente es una adaptación del cliente de minas que se desarrolló en [Echeva-1], por lo que si el lector desea conocer en profundidad su funcionamiento deberá remitirse a aquel artículo. En cualquier caso, no es necesario conocer su funcionamiento para realizar su adaptación al mundo WWW. Este es un factor importante, ya que demuestra que es posible migrar aplicaciones al WWW si se dispone de los conocimientos necesarios en este último ámbito, aunque no se cuente con ningún conocimiento en las áreas de que trata el programa. Así, si se puede realizar una partición de la aplicación en función de sus puntos de E/S, bastará con cambiar estos puntos sin necesidad de tocar el resto del programa. Por ejemplo, donde antes decía:

```
if ((x==1) && (y==0))
{
  printf("Enviada senal de exterminio al servidor\n");
  exit(0);
}
```

ahora se indica:

```
if ((x==1) && (y==0))
{
  printf("Content-type: text/html\n\n");
  printf("<H1>WEB MINES</H1>\n");
  printf("The game's over but... I'll be back<p>\n");
  printf("<A HREF=/index.html>\n");
  printf("<IMG SRC=/httpd-internal-icons/back.xbm
        ALT=Echeva's Server>\n");
  printf("Echeva's Server\n");
  printf("</A><p>");
  exit(0);
}
```

Si se separan las funciones DE E/S en módulos independientes puede ser posible portar una aplicación a WWW tocando únicamente estos módulos

En el disco de la revista se adjuntan tanto la versión WWW del juego como la versión para terminales de texto que ya se adjuntó con el número 18. De esta forma, el lector podrá comparar las aplicaciones cliente y servidor por sí mismo para estudiar sobre las fuentes los cambios realizados para su adaptación al WWW. En el ejemplo facilitado, el principal cambio ha sido la sustitución del procedimiento *mostrar* por el procedimiento *MostrarHTML*, ambos encargados de mostrar el tablero en pantalla. De esta forma la presentación del tablero, que antes se realizaba mediante:

```
if (buf[0] == 'F') mostrar(0); else mostrar(1);
```

ahora ha pasado a ser:

```
if (buf[0] == 'F') MostrarHTML(0); else MostrarHTML(1);
```

lo que lleva a pensar que bastaría haber realizado dos módulos que implementaran la funcion *mostrar()* de formas diferentes para, mediante una compilación condicional y sin necesidad de realizar ninguna modificación en el cliente, disponer de dos versiones del mismo, una para terminales de texto y otra para WWW. Por ello, conviene siempre dejar aisladas las funciones de E/S.

CONCLUSION

La aplicación comentada en este artículo presenta todas las dificultades y todas las técnicas y métodos que se pueden presentar en programación de aplicaciones WWW basadas en el estandar CGI. Con ello se puede considerar finalizada la serie dedicada a este estándar, aunque aún quedan algunos pequeños detalles por tocar, como el envío de ficheros a través de *forms*. Por ello, medite el lector sobre el código y plantee otros diseños y formas alternativas de realizar la aplicación, así como formas de mejorarla. Por ejemplo, observe el código correspondiente al servidor y considere si no se producirá algún problema si dos servidores tratan de modificar la tabla índice al revés. ¿Se podría evitar generar un enlace por cada casilla del tablero? Y para aquellos que sigan la sección de sistemas abiertos, ¿cómo se podría mejorar el protocolo de comunicación entre cliente y servidor?

SOFTWARE FACILITADO

Junto con la revista se facilitan los *ficheros sp-minas.tgz* y *mines092.tgz* con los fuentes para Linux de las aplicaciones descritas. El primero de los ficheros corresponde a la aplicación cliente-servidor de buscaminas que se entregó con el número 18 de la revista. El segundo fichero corresponde a la adaptación para funcionar como aplicación WWW. Esta última aplicación hace referencia a las imágenes de las casillas, que deben encontrarse en el directorio */minas* del servidor (o cambiarse en las fuentes). No se facilitan imágenes con la aplicación, por lo que el lector deberá diseñar las suyas.

REFERENCIAS

Todas las referencias aluden a otros artículos del mismo autor publicados en Sólo Programadores según:

[Echeva-1], "Arquitecturas cliente-servidor: La interfaz socket II", núm. 18
[Echeva-2], "La interfaz CGI: primer contacto", núm. 20
[Echeva-3], "CGI: Metodología y transmisión de estado", núm. 28
[Echeva-4], "La interfaz CGI: descripción avanzada", núm. 25
[Echeva-5], "Arquitecturas cliente-servidor: La interfaz *socket*", núm. 17

CONTACTAR CON EL AUTOR

Para cualquier duda, comentario, sugerencia o crítica, se anima al lector a ponerse en contacto con el autor mediante:

E-mail Internet: echeva@dit.upm.es
E-mail Compuserve: 100646,2456
http://highland.dit.upm.es:8000

INTRODUCCIÓN AL LENGUAJE DE PROGRAMACIÓN ADA

Felipe Bertran y César Sánchez

Con el presente artículo se da comienzo a una serie sobre el lenguaje de programación Ada. Este mes se explicarán los precedentes y algunas de las principales características del lenguaje, que se irán ampliando en los posteriores artículos de la serie. También se explicará el método de compilación y se realizará el primer programa Ada.

El nombre de Ada fue adoptado en honor de Augusta Ada Byron (1815-52), Condesa de Lovelace, hija del poeta Lord Byron. Ada puede ser considerada como la primera persona que programó una máquina. Fue como ayudante de Charles Babagge en el uso de su máquina analítica mecánica. El nombre Ada, por lo tanto, no es un acrónimo, y por ello debe escribirse Ada y no ADA.

Ada, como lenguaje de programación, tiene sus orígenes en 1974, cuando el departamento de defensa de los Estados Unidos decidió financiar su creación. Las motivaciones fueron los fuertes costes que suponían los proyectos de software, en especial en el área de sistemas empotrados. Éstos son sistemas que forman parte de un sistema físico mayor, los que, por ejemplo, gobiernan un avión de combate, dirigen un misil, o controlan un horno microondas.

Estos costes tan elevados se debían, fundamentalmente, a la dificultad de coordinar multitud de equipos de desarrollo diferentes que, entre todos ellos, empleaban decenas de lenguajes de programación distintos y a la dificultad de depurar y sobre todo mantener el código existente.

Aún más, eran también muy elevados los costes de formación y la inversión en herramientas, compiladores..., para toda esta variedad de lenguajes y dialectos.

En este contexto, se decidió que era necesario establecer estándares, sobre todo en el citado marco de los sistemas empotrados. El resultado fue el lenguaje Ada, estandarizado en 1983, y por ello conocido como Ada83.

Como ventajas de ser el resultado de un proceso de estandarización se puede considerar el que todos los fabricantes de compiladores, todos los grupos de programadores y todos los libros existentes hablen de un mismo punto sin ambigüedad alguna. Sólo existe libre interpretación donde el estándar explícitamente lo permita y en las condiciones en que lo haga. También se cuenta con una única referencia, definición oficial, en el documento conocido como "Manual de Referencia del Lenguaje de Programación Ada", a cuyas secciones se refieren con frecuencia los errores que muestran los compiladores, los documentos especializados,...

A partir de 1988, y durante los primeros años de la decada de los 90, tuvo lugar una revisión del estándar. El lenguaje Ada había sido utilizado en muchos ámbitos fuera de los sistemas empotrados, y nuevos avances tecnológicos y paradigmas cada vez más aceptados debían ser soportados. El resultado fue el nuevo estándar de Ada de 1995, con lo que el lenguaje paso a ser conocido como Ada95, o simplemente Ada (empezó a usarse entonces el término Ada83 para referirse al lenguaje antes de la revisión). Se trató de conservar la compatibilidad hacia atrás, con lo que la inmensa mayoría de programas realizados usando Ada83 pueden ser mantenidos y extendidos directamente con los compiladores y herramientas de Ada (entiéndase Ada95).

Los campos en los que se hizo más hincapié en la revisión fueron la programación orientada a objetos, ahora

soportada en toda su plenitud, y la programación en tiempo real, campo en el que Ada ya era pionero y ampliamente utilizado. De estos temas se hablará en posteriores artículos.

Hoy en día, todos los programas que se escriban para el departamento de defensa de los Estados Unidos deben estar escritos en Ada. Además, Ada ha sido empleado con éxito en otros muchos proyectos civiles, tales como sistema de control de tráfico aéreo, las líneas del tren de alta velocidad francés (TGV), proyectos de biomedicina, proyectos de la Agencia Espacial Europea, etc...

CARACTERISTICAS GENERALES

Puede pensarse en Ada como en un lenguaje extenso, que soporta muchos paradigmas. Es un lenguaje muy útil para la ingeniería de software, en la realización de proyectos de gran tamaño. No en vano, Ada se definió pensando en la fiabilidad, la portabilidad, la modularidad, la reutilización de código, el mantenimiento y la eficiencia. En este sentido, se irá comprobando cómo Ada obliga a una cierta disciplina que hace el código más organizado. Se comprenderá por qué se suele decir que Ada es más que otro lenguaje de programación.

Ada es, por lo tanto, un lenguaje de programación con soporte completo de orientación a objetos, y como tal, permite la creación de tipos abstractos de datos (que, someramente, son estructuras de datos a las que se añaden ciertas operaciones que permiten su manipulación), encapsulación, polimorfismo, herencia y enlazado dinámico.

Otro punto fuerte de Ada ligado a la programación de sistemas de tiempo real es la gestión de programas concurrentes (programas que tienen varios flujos de ejecución -tareas- funcionando al mismo tiempo). Ada proporciona sistemas de sincronización entre los flujos y permite definir objetos protegidos para evitar que los datos puedan ser manipulados peligrosamente por varias tareas al mismo tiempo.

Ada es un lenguaje de programación con soporte completo de orientación a objetos

El presente curso se estructura en tres ciclos. El primero se dedica a estudiar los aspectos básicos del lenguaje: los tipos de datos, una introducción a la estructuración de los programas (en paquetes, funciones y procedimientos), unas nociones sobre el control del flujo de ejecución (bucles, alternativas), declaraciones y los operadores básicos.

En el segundo y tercer ciclo se estudiarán y se profundizará en otros aspectos más avanzados del lenguaje, como excepciones, encapsulación, paquetes genéricos, concurrencia, desarrollo de librerías jerárquicas y programación orientada a objetos. Ada es un lenguaje que soporta muchos más paradigmas que Pascal o C y, por lo tanto, se tarda más tiempo en llegar a dominarlos todos ellos. El coste de tener un lenguaje tan flexible es la complejidad del mismo. Ada, a pesar de esta aparente dificultad, presenta varias ventajas.

En primer lugar, no es necesario conocer todas las posibilidades del lenguaje para escribir programas perfectamente aceptables y suficientemente potentes. Puede decirse que muchas de sus características son ortogonales, independientes. En otros lenguajes es necesario tener un conocimiento bastante profundo para escribir incluso programas relativamente sencillos, ya que si no se puede incurrir en errores graves que provoquen fallos inesperados por desconocidos. Mucha gente de la comunidad de tiempo real, por ejemplo, utiliza Ada95 sin aprovechar la programación orientada a objetos en toda su plenitud.

En segundo lugar, una vez conocido, los programas son mas sencillos de escribir y leer, más sencillos de depurar y, sobre todo, más sencillos de entender una vez que ha pasado un cierto tiempo desde que se escribieron. También, como ya se ha mencionado antes, las características propias de Ada hacen que los programas sean susceptibles de tener menor número de errores.

LISTADO 1

```
with Text_IO;
   procedure Hello is
   begin
      Text_IO.Put_Line("Hello World!");
   end Hello;
```

Son estas características las que han llevado a muchas voces autorizadas a recomendar en entornos educativos Ada como primer lenguaje de programación. Se aprende un lenguaje industrialmente muy potente haciendo programas desde el principio.

Para seguir óptimamente la serie es conveniente instalar en el ordenador un compilador de Ada. La mejor opción es, sin duda, el compilador GNAT (*GNU and New York University Ada Translator*), que además de ser un compilador de calidad, tiene la ventaja de ser de libre distribución y estar disponible para varios sistemas operativos. No debe confundirse libre distribución con gratis. La realización del compilador costó en su momento al estamento público americano algo más de 3 millones de dólares. Aun así, se considera que el hecho de tener un compilador de calidad para la comunidad científica e industrial compensa esta inversión. Muchos programas se dejan en Internet con el código fuente disponible y licencias que permiten copiarlo y modificarlo para volver a distribuirlo. Linux es otro ejemplo de ello. Estos programas consiguen alcanzar cotas de calidad, ya que hay muchos usuarios que lo modifican y redistribuyen, algo que sería imposible con otras licencias de distribución.

El compilador GNAT se incluye en el CDROM de la revista, en sus versiones para MsDOS y para Linux.

EL PRIMER PROGRAMA. HELLO WORLD!

Tras la introducción histórica y de características se presenta el primer programa en Ada. Se trata del popular "Hello World", un pequeño programa que imprimirá en la pantalla ese texto como mensaje de bienvenida. Con ello se pretende habituar al lector a la sintaxis del lenguaje y al procedimiento de

compilación y enlazado con GNAT. El programa se muestra en el listado 1.

A continuación se analiza el código. Puede observarse que se ha escrito un procedimiento llamado *Hello*. Las instrucciones que han de ejecutarse para completar este procedimiento se encuentran entre "*begin*" y "*end Hello;*". Todas las sentencias en Ada terminan en punto y coma. Las líneas del programa que no terminen en punto y coma no son sentencias. Como puede verse, el procedimiento *Hello* consta tan sólo de una sentencia y del *end*. La sentencia *end* puede seguirse, opcionalmente, por el nombre del procedimiento o función que termina. Se recomienda habituarse a hacerlo así para ayudar al compilador a detectar y mostrar posibles errores.

La línea que comienza por la clausula "*with*" indica que se usará en las instrucciones a continuación algo del paquete llamado *Text_IO*. Este tipo de cláusulas se llama "cláusula de contexto" (*context clause*) porque especifica el contexto en el que el procedimiento *Hello* se va a compilar.

Text_IO es un "paquete" estándar que viene con el lenguaje Ada. Un paquete puede verse como un almacén de código que podemos utilizar. En otros lenguajes existen conceptos similares que comunmente se llaman bibliotecas (*libraries*). Entre otras cosas, el paquete *Text_IO* contiene un procedimiento llamado *Put_Line* que puede aceptar un parámetro de tipo *String* (una cadena alfanumerica, que en nuestro ejemplo es "*Hello World*"), y que como efecto lo presenta en pantalla. La llamada al procedimiento *Put_line* de *Text_IO* ha tenido que indicarse precedida del nombre del paquete separados por un punto.

En resumen, la lectura de este simple programa en lenguaje natural podria ser:

"Voy a usar algo de *Text_IO*. A continuación, te muestro cómo hacer *Hello*: llama a *Put_Line*, que está dentro de *Text_IO*, diciéndole que use la cadena "Hello_world!". Terminé de describir Hello."

Existe una manera para no tener que especificar el nombre del paquete en que se encuentra un determinado procedimiento en las llamadas a él. Esto se consigue mediante el uso de la clausula *use*. Así, el ejemplo anterior podría reescribirse añadiendo esta cláusula, según se observa en el listado 2.

Con la cláusula *use* se le pide al compilador que complete él mismo las llamadas a los procedimientos en las que no esté especificado el paquete al que pertenecen. Sin embargo, se recomienda no abusar de esta práctica, puesto que el código, aunque más corto, puede resultar menos claro.

Con GNAT se dispone de un compilador de Ada de calidad

Si se necesitara usar procedimientos de varios paquetes, se escribirían varias cláusulas *with*, una para cada paquete. También se pueden especificar varias cláusulas *se*, hasta una por paquete que se haya incluido con *with*.

El uso de varias clausulas *use* podría provocar algun conflicto por encontrar el compilador varios procedimientos o funciones con el mismo nombre en varios paquetes (aunque,como se vera más adelante, el compilador será capaz de distinguir en la mayoría de los casos). Esto podría solucionarse utilizando la notación que se usaba sin el *use* (la de la llamada a *Put_Line* del listado 1) para esa llamada. La utilización de *use* permite, pero no obliga, la simplificación de la llamada.

En Ada no se distingue, como en otros lenguajes (por ejemplo C), entre programa principal y subprogramas. Un procedimiento puede ser llamado desde otro procedimiento o ser el programa principal. En nuestro ejemplo, el procedimiento *Hello* será el programa principal.

COMPILACIÓN

El modelo de compilación de Ada es muy similar al de la mayoría de los lenguajes compilados modernos. Se describe a continuación el método a seguir para compilar los sencillos programas de ejemplo con GNAT.

LISTADO 2

```
with Text_IO;
use Text_IO;
procedure Hello is
begin
    Put_Line("Hello World!");
end Hello;
```

El código Ada en texto correspondiente a un programa se denomina código fuente del programa. Este código fuente es introducido en una herramienta, denominada compilador, que genera, si el programa es correcto, un fichero de datos binario cuyo contenido se denomina código objeto. Este código debe ser introducido a su vez en una segunda etapa en la que se enlaza junto a otras piezas, otros códigos objetos, para generar código ejecutable.

El programa se puede escribir con cualquier editor ASCII (por ejemplo con el editor de MSDOS "EDIT"). Cabe señalar que en Ada no se distingue entre mayúsculas y minúsculas. Los programas anteriores serían igualmente válidos si se hubiese escrito, en vez de *Procedure*, *procedure* (o incluso *PrOcEdURE*). Nótese que esto también se aplica para los identificadores, y así el procedimiento *Hello* sería el mismo que el *hellO* (incluso si se pusiese de diferente manera al principio que en el *end*).

Puede considerarse Ada como un lenguaje fuertemente tipado

En esta serie se seguirá el convenio de escribir todas las palabras reservadas del lenguaje (*begin*, *end*, *procedure*, *is*...) en minúscula y los identificadores (como *Hello*, *Put_Line*, o los nombres de las variables) con la primera letra en mayúscula. La indentación tampoco es significativa ni relevante para el compilador, aunque se recomienda seguir el estilo de los listados presentados.

Una vez escrito, se grabará el listado en un fichero que se llamará, por ejemplo, HELLO.ADB, y luego se compilara

con el compilador de Ada de GNU (GNAT, presentado anteriormente), escribiendo desde la línea de órdenes de MsDOS lo siguiente:

gcc -c HELLO.ADB

Esto generará dos ficheros: HELLO.O y HELLO.ALI. Para obtener el programa ejecutable todavía queda enlazar el programa con unas librerías necesarias que proporciona el compilador. También debe especificarse qué procedimiento será el procedimiento principal. Sería posible, como se indicó antes, que otro procedimiento utilizase a *Hello*. Todo ello se hace con la orden:

gnatbl HELLO.ALI

Con ello se genera, finalmente, el fichero ejecutable HELLO.EXE.

SEGUNDO PROGRAMA. INTRODUCCION A LAS VARIABLES

Una variable tiene en Ada el sentido clásico en programación. Las variables son zonas de memoria en las que se almacenan valores. Las variables tienen un nombre que comúnmente se denomina identificador de la variable. El valor asociado a la variable, que es el que está almacenado en la celda de memoria asignada a ella, es conocido como valor de la variable. Se denominan variables porque los valores pueden cambiar.

El significado del contenido de una variable puede ser muy diferente.

Puede representar un número, una cadena de caracteres o una ficha de una base de datos, o incluso una estructura compleja de estas fichas. A lo largo de la serie se aprenderá a escribir nuestros propios tipos de variables y a utilizar los ya creados.

Ada puede ser clasificado como un lenguaje fuertemente tipado. Esto quiere decir que ninguna variable asociada a un tipo podrá ser rellenada con información correspondiente a otro tipo o se podrán establecer comparaciones entre objectos de distintos tipos. Ni siquiera por error (esto es precisamente lo que se consigue evitar automáticamente).

Las variables en Ada necesitan ser declaradas previamente. Con ello se indica al compilador que esa variable podrá ser utilizada, y que tipo le corresponde. En el listado 3 se presenta un ejemplo de la declaración local de tres variables: A, B y C. Las sentencias en las que se declaran se denominan sentencias declarativas. Como puede observarse, aparecen dentro del procedimiento, pero antes del "*begin*" que marca el comienzo de las instrucciones de código.

Los comentarios no generan código ejecutable y sirven para hacer más inteligible el programa

Cuando se llame al procedimiento "*Ejemplo*", lo primero que ocurre es que se "elaboran" las sentencias declarativas. Entonces se creará el espacio en memoria necesario para almacenar las tres variables, espacio que existirá hasta que se alcance la sentencia "*end Hello;*". Opcionalmente se inicializarán las variables en ese momento, si se ha indicado. Después de la sentencia "*end*", la zona de memoria reservada se libera y las variables A, B y C desaparecen y no pueden ser referenciadas. Por ello se llaman variables locales: no existen fuera del procedimiento "*Ejemplo*".

Además, en el caso de que se vuelva a llamar al procedimiento "*Ejemplo*", el contenido de las variables locales no se conserva, y el procediemiento de elaboración descrito volvería a comenzar.

Puede comprobarse que A y B son inicializados en la declaración, es decir, se les asigna un valor a la hora de declararse. Éste será el valor que tendrán antes de empezarse a ejecutar la primera instrucción. Se puede observar también que B es en realidad una constante. Su valor asociado no podrá ser modificado. Además, la inicialización en este caso es obligatoria.

LISTADO 3

```
with Text_IO;
procedure Ejemplo is
    A  :  Integer :=3;
    B  :  constant Integer := 10;
    C  :  Float;
begin                    —Esto es un comentario
    Text_IO.Put_Line("Esto es un ejemplo.");
end Ejemplo;
```

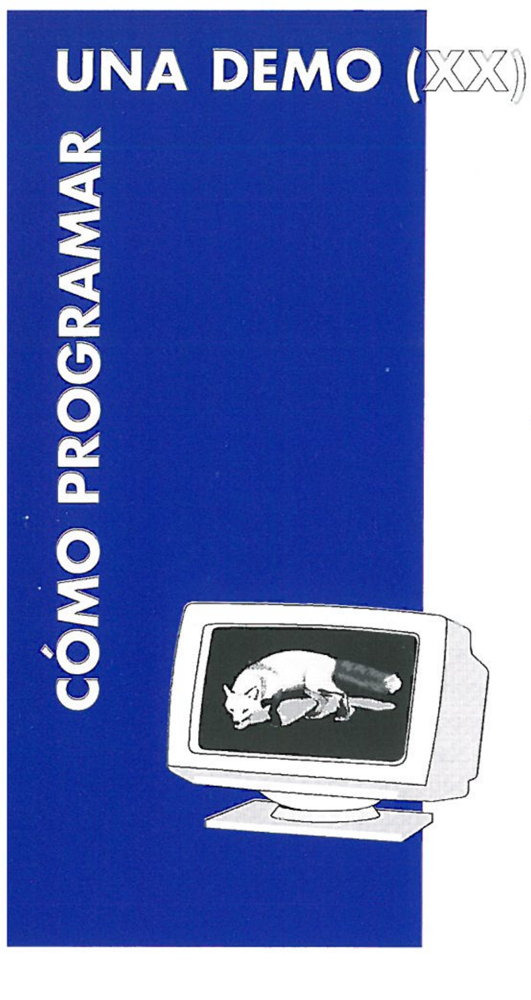
En el caso de que dentro del procedimiento *Ejemplo* existiera alguna sentencia que pudiera modificar el contenido de B, el compilador nos avisaría generando un error.

Puede observarse que en el listado 3, detras de la palabra *begin*, aparece un comentario. Los comentarios no generan código ejecutable y sirven para hacer más inteligible el programa. Deben ir precedidos de un doble signo menos y se extienden hasta el final de la línea, independientemente de su contenido. En los programas escritos en Ada suelen ser necesarios menos comentarios que en los programas escritos en otros lenguajes, debido a que la sintaxis propia de Ada hace el código más "autoexplicativo".

Los identificadores (de momento, los nombres de las variables y de los procedimientos) pueden ser tan largos como se quiera, hasta la longitud de una línea. Pueden contener exclusivamente dígitos, letras y subrayados (_), aunque siempre deben empezar con una letra. Los subrayados siempre deben seguirse de una letra o un dígito, por lo que no pueden ir dos subrayados seguidos ni se puede terminar un identificador con un subrayado.

RESUMEN

Se han descrito someramente algunas de las características del lenguaje Ada, y se ha escrito y compilado el primer programa. Con él se ha aprendido a utilizar paquetes ya escritos. Más adelante se mostrará cómo escribir programas nuevos o extender los ya existentes. También se ha mostrado cómo escribir sentencias que llamen a otros procedimientos y cómo declarar e inicializar variables y constantes locales. Se recomienda probar el proceso básico de compilación descrito con los ejemplos.



EL COMPILADOR TMT PASCAL

Pedro Antón Alonso

El motivo por el que no se ha optado por un compilador de C es continuar con la tónica de aprendizaje que se ha llevado a lo largo de este curso. Aquellos lectores que habitualmente programen en C no tendrán ningún tipo de problema para traducir el código del curso a dicho lenguaje. Algo, por otro lado, mas complicado de hacer al contrario. Qué duda cabe de que el Pascal es altamente didáctico.

El chequeo estricto de variables hará que el código generado por el compilador sea más estable

COMENZANDO CON EL COMPILADOR

TMT Pascal para DOS es un compilador de 32 bits para DOS en desarrollo, del cual se dispone de una versión de evaluación totalmente gratuita. La restricción de esta versión es el tamaño del programa a ejecutar, estando éste limitado a aproximadamente 1.5 Mb incluyendo código, datos y pila. Para crear un ejecutable de mayor tamaño se debe obtener la licencia. El compilador requiere 2 Mb libres de memoria extendida y coprocesador matemático.

INSTALANDO EL COMPILADOR

Como el compilador viene empaquetado en un fichero tipo ZIP, lo primero que se debe hacer es desempaquetarlo convenientemente, es decir, con la opción de incluir subdirectorios, en la parte del disco duro que nos interese. Esta operación creará 3 ficheros de documentación y licencia, así como los siguientes subdirectorios:

- *BIN*. Este subdirectorio contiene los ficheros ejecutables del compilador, así como el fichero de configuración.
- *UNITS*. En este subdirectorio se incluyen las unidades desarrolladas para este compilador. Con ellas se consigue una compatibilidad casi total con el Pascal de Borland.
- *EXAMPLES*. Como indica su nombre, el subdirectorio contiene varios ejemplos que se suministran con el compilador de TMT Pascal.

Estamos en la época de los procesadores de 32 bits, y cada vez se hace más necesario un compilador de este tipo. TMT Pascal permite usar, con pequeñas variaciones, el código que se ha ido generando a lo largo de este curso, pero con toda la potencia de los 32 bits.



Figura 1

- *DOS32*. Este subdirectorio contiene el extensor de 32 bits para DOS *DOS32*, tal y como se incluye para su uso y distribución. Este extensor se usó en un principio para el desarrollo del compilador y posteriormente fue sustituido por una variación del *PMODE* de *Tran*.

Una vez que el subdirectorio *BIN* pase por el *path* del sistema operativo, sólo resta configurar el compilador por mediación del archivo *PLT.CFG* los parámetros que acepta el fichero de configuración, que son los siguientes:

w+ Mensajes de alerta activados.

r+ Chequeo de rangos activo.

q+ Chequeo de sobrepasamiento (*overflow*) activo.

i+ Chequeo automático de la operaciones de entrada /salida.

t+ Tipo puntero.

x+ Sintaxis extendida.

v+ Chequeo de variables estricto.

optreg+ Optimización por registro.

optfrm+ Optimización por pila.

opt+ Ambas optimizaciones (optreg+ y optfrm+).

Los parámetros por defecto son: w+, r+, q+, i+, t-, v+, x+, opt+.

Destacar de entre ellos el chequeo estricto de variables. Aunque sea algo a lo que no se esté acostumbrado y se haga un poco duro de trabajar en principio, hará que el código generado por el compilador sea más estable.

La especificación de los *buffers* se indica con los parámetros siguientes:

- *objmax <tamaño>* Indica el tamaño del *buffer* de compilación. Debe tener un valor superior en la mitad al tamaño máximo del mayor fichero *OBJ* del programa. Nótese que los ficheros *OBJ* del compilador tienen extensión *FPD*.
- *exemax <tamaño>* Indica el tamaño máximo del ejecutable a crear.

En modo protegido, los segmentos no son usados, por lo que no deben ser usados los registros que los referencian

Los valores por defecto de estos parámetros son: *objmax 333000* y *exemax 256000*.

La especificación de tamaño de la pila o *stack* se realiza con:

- *stack <tamaño>* Indica el tamaño de la pila a usar en la aplicación. El valor que toma por defecto es de 32000 bytes.

La especificación de localización de ficheros viene dada por los siguientes parámetros:

- *srcpath <unidad:subdirectorio>* Indica el camino donde se encuentra el código fuente a compilar.
- *objpath <unidad:subdirectorio>* Indica el subdirectorio donde se encuentran los ficheros con extensión *FPD* (unidades) y los ficheros de soporte del extensor de 32 bits para DOS. Debe ser aquel donde se encuentren los ficheros que se descomprimen en *UNITS*.
- *objimppath <unidad:subdirectorio>* indica dónde se encuentran aquellos ficheros con extensión *OBJ* que serán usados con la directiva $L del compilador, tal y como ocurría con los compiladores de Borland.

Nombre del extensor de 32 bits a usar:

- *stub <nombre>* Indica el nombre del extensor a emplear. Por defecto el nombre es PASSTUB.EXE, fichero que debe encontrarse en el mismo subdirectorio que se las unidades, es decir, en aquel especificado por el parámetro *objpath*.

ORGANIZACIÓN DE LA MEMORIA

En modo protegido no existen los segmentos.

El compilador usa una variación del extensor de *Tran* (personaje muy conocido en el mundo de la *demoscene*), el *PMODE*, compatible en gran parte con el extensor *DOS32* para generar aplicaciones en modo protegido.

Se deben tener muy en cuenta que, en modo protegido, los segmentos no son usados, y por lo tanto no deben ser usados los registros que los referencian. La memoria se organiza en páginas de 4 Kb. La dirección física de estas es almacenada en una tabla, de tal forma que la dirección lógica no necesariamente coincidirá con la dirección física. Unicamente con el primer mega de memoria se puede asegurar que la dirección de memoria física coincide con la lógica. De esta forma se puede acceder sin problemas y directamente a las zonas de memoria ubicadas en esta zona, por ejemplo, la memoria de vídeo.

PASO DE PARAMETROS

Los parámetros se pasan por la pila.

Al igual que ocurre con los compiladores de Borland, los parámetros de entrada a una función son pasados a ésta a través de la pila. Aunque, como cabía esperar, existe alguna diferencia: los parámetros usan siempre 4 bytes de la pila.

Cuando se generen funciones en ensamblador, bien sea en línea o en un fichero externo, se ha de tener siempre bien en cuenta que se debe preservar el contenido de los registros EBX, ECX, EDX, ES y DS. Los registros de segmento, como ya se ha comentado, no deben ser tocados nunca bajo ningún concepto. Como ocurre en Borland Pascal con el registro AX, el resultado de una función se devuelve en este caso en el registro EAX.

SINTAXIS DEL COMPILADOR

TMT Pascal para DOS es, generalmente, compatible con el Pascal de Borland.

Aunque se puede usar prácticamente el mismo código para ambos compiladores, se deben tener en cuenta algunas variaciones que existen entre ambos lenguajes.

En general, TMT Pascal es más estricto que Borland en lo que se refiere a la asignación de variables de distinto tipo.

TMT Pascal permite salir de un procedimiento local a otro. Algo que per-

mite realizar el Pascal ANSI, pero no permite hacer Borland. Hay que tener en cuenta que, para el compilador, traducir una instrucción de salto, bien sea condicional o incondicional, a ensamblador es relativamente sencillo y, por otra parte, puede resultar extremadamente útil al programador en determinadas ocasiones. Aunque con ello no se genere un código elegante, se puede generar un código más efectivo.

Sin embargo, no es posible salir de un procedimiento usando las instrucciones *BREAK* o *CONTINUE* en TMT Pascal. Para realizar esta operación se debe usar *GOTO*.

imposible, por lo que se recurría al pequeño truco de escribir dichas instrucciones en hexadecimal. Generalmente sencillo, puesto que la mayoría de las instrucciones en ensamblador del modo protegido son idénticas a las del modo real si a ésta ultima se le antepone un 66h. Es decir, *mov ax,0000h* se corresponde con el valor hexadecimal B80000h, y la instrucción *mov eax,00000000h* se corresponde con el valor hexadecimal 66B800000000h. En modo protegido esto mismo sucede al revés, es decir, el valor hexadecimal B8 se corresponde con el operando *mov eax* y el valor hexadecimal 66B8 con *mov ax*, algo que deberá tenerse muy en cuenta. Aunque la gran ventaja de construir código ensamblador bajo TMT Pascal se obtiene en la ausencia de segmentos y en la total libertad para usar las instrucciones ensamblador en modo protegido. En el listado 1 se observa cómo queda modificado el código en ensamblador que escribe un pixel en la zona de memoria indicada.

Además, se ha implementado la instrucción *CODE*, cuyo funcionamiento es análogo a la instrucción *ASSEMBLER*, con la salvedad de que evita las cabeceras de entrada y salida de función, incluso el *RET*, consiguiendo de esta forma mayor control sobre el código ensamblador escrito.

TMT Pascal para DOS es, generalmente, compatible con el Pascal de Borland

El programa principal puede tener una parte estructurada como si de una unidad se tratara, es decir, otros ficheros de código pueden incluir el programa principal como si de una unidad se tratase, accediendo a aquellas funciones y procedimientos declarados en la parte *INTERFACE*. Esto implica, como es de esperar, que el nombre del fichero y el de la instrucción *PROGRAM* siempre deben ser iguales.

Como era de esperar, al ser TMT un compilador de modo protegido, se ha creado un nuevo tipo de variable, *DWORD*, que es equivalente al tipo *LONGINT* (variable entera de 4 bytes con signo) sin signo. Este tipo de variable es especialmente útil al acceder a direcciones de memoria, pero se ha de tener un especial cuidado con su uso ya que, en operaciones aritméticas entre argumentos de ambos tipos, las variables DWORD son convertidas a LONGINT. Conversión que podría producir algún error de cálculo.

ESCRIBIENDO FUNCIONES EN ENSAMBLADOR

La construcción de funciones en ensamblador por medio de la instrucción *assembler* es algo muy usado en el código que se ha ido desarrollando a lo largo de este curso. Pero recordemos que usar una instrucción de 32 bits en una función escrita en Borland era

LA UNIDAD DOS

La unidad DOS que se suministra con el compilador merece una especial atención.

Aunque el manejo de las unidades que se suministran con el compilador viene documentado en el fichero MANUAL.DOS, también incluido en el compilador y fichero en el que está basado gran parte de este artículo, la unidad DOS es la que más puede llamar la atención del lector.

Básicamente, esta unidad se corresponde con la unidad DOS de Borland, pero como se está hablando de un compilador en modo protegido, hay unos cuantos detalles que se deben tener muy en cuenta.

Se ha creado un nuevo tipo de datos denominado *FarPointer* de estructura:

```
type
  FarPointer = record
    ofs: pointer;
    seg: word
  end;
```

Este tipo de datos es el pasado a las funciones *GetIntVec* y *SetIntVec*, en lugar del tipo *pointer* que pasaba Borland.

También varía la cabecera de un procedimiento de tipo *interrupt*. Ahora dicha cabecera debe tener la siguiente estructura:

PROCEDURE MyHandler (eip, eax, ecx, edx, ebx, esp, ebp, esi, edi: dword; gs, fs, es: word); interrupt;

Recordemos que se trata de un compilador todavía en desarrollo y se debe investigar con este tipo de nuevas estructuras antes de decidir usarlas.

Otro tipo nuevo que añade la unidad DOS, y que también llama la atención, es el tipo *registers*. La nueva estructura es:

```
type
  Registers = record
    edi, esi, ebp, _res : dword;
    case boolean of
     true:  (ebx, edx, ecx, eax: dword;
             flags, es, ds, fs, gs, ip, cs,
sp, ss: word);
     false: (bl, bh, b1, b2, dl, dh, d1,
d2,
             cl, ch, c1, c2, al, ah: byte);
    end;
  end;
```

Recordemos que únicamente se corresponde la paginación lógica con la física en el primer mega de memoria. Por lo tanto, si se desea usar el uso de las funciones *Intr* y *MsDos* esto es algo a preveer, ya que el MSDOS sólo podrá trabajar correctamente con ese mega.

Y para comenzar a comentar el código que acompaña a este artículo, finalizaremos con otra diferencia importante existente entre la programación en modo real y en modo protegido. Como se ha comentado sucesivas veces a lo



largo de este artículo, trabajar en modo protegido implica la ausencia de segmentos. Evidentemente la función *seg* de Borland devolverá siempre un valor cero. Por otro lado, no será necesario el uso de dicha función, al devolver la función *ofs* un valor de tipo *dword* con el que se puede direccionar de forma lineal hasta 4 Gb de memoria.

en coma flotante y haciendo uso exhaustivo del coprocesador matemático. Esta unidad no ha sido incluida en este artículo debido a los inesperados resultados obtenidos en lo que se refiere a generación de código usando la coma flotante por este compilador.

A continuación se comentará el sustancial cambio que se ha producido en la librería: la definición de los tipos a usar. El anterior diseño de esta unidad cerraba mucho las posibilidades de desarrollo y ampliación tanto de funciones como de estructuras. Por ello se ha optado por trabajar con tipos y punteros. De esta forma, se abre una puerta a las posibles ampliaciones del mundo del diseño tridimensional. Trabajar con punteros es algo que ralentiza el proceso de acceso a los datos, ya que son necesarios más cálculos para acceder a los datos, pero facilitará enormemente las cosas a la hora de generar aplicaciones paralelas que permitan cargar o grabar las estructuras de datos generadas o modificadas por nuestro código. Pero no sólo eso. Trabajar con estructuras dinámicas de datos abrirá la posibilidad de ir incluyendo las estructuras que definen los objetos a modelar, nuevos campos, tales como vector normal de un punto o vector normal de una cara, algo absolutamente necesario para continuar con la parte de tres dimensiones del curso.

La notación de tipos que se va a emplear es comúnmente usada por los compiladores. No obstante, pasaremos a referenciarla:

El prefijo *T* delante del nombre de una variable indicará que se esta refiriendo a un tipo y el prefijo *P* a un puntero. De esta forma la variable *TPoint3D* hace referencia, evidentemente, al tipo *Point3D*, que por otra parte es una estructura de datos que, de momento, contiene los campos x,y,z. Así pues la variable *PPoint3D*, será un puntero a una variable de tipo *TPoint3D*. En el listado 4 se pueden observar los tipos creados para trabajar con la librería.

Trabajar con memoria dinámica obliga a reservar espacio para cada estructura de datos con la que se desee trabajar. Y no hay que olvidar liberar dicho espacio cuando se hay finalizado de trabajar con él. El lector podrá observar un claro ejemplo de esto en el código DONUTFP.PAS que acompaña a este artículo.

DWORD es un nuevo tipo de variable equivalente al tipo LONGINT (variable entera de 4 bytes con signo) sin signo

LA UNIDAD MCGA

La unidad DEMOVGA cambia de nombre y se actualiza.

Esta unidad se ha llamado MCGA para la versión de TMT Pascal, aunque contiene las mismas funciones, eso si, perfectamente adaptadas al nuevo compilador. Esto implica modificar las variables por defecto a variables de 4 bytes, que son mas rápidamente accedidas por un compilador de 32 bits (no hay que olvidar que ahora los registros por defecto son de 4 bytes), así como la total supresión de los segmentos en todo el código.

En el listado 2 se muestra cómo quedan las funciones de manejo de memoria de la unidad MCGA con el nuevo compilador, y en listado 3 se puede ver cómo eran dichas funciones para el compilador de Borland. Observando ambos listados se aprecia claramente cómo ha evolucionado el código de un compilador al otro.

LA UNIDAD LIB3DFP

La unidad LIB3D se ha modificado sustancialmente.

Con el cambio del compilador, prácticamente se ha reescrito todo el código del curso. Aprovechando esta circunstancia se ha modificado la estructura de los datos y las funciones de la unidad LIB3D.

En primer lugar, las letras FP al final del antiguo nombre indican que esta unidad trabaja en punto fijo (*Fixed Point*), dejando abierta la posibilidad al desarrollo de una nueva librería de tres dimensiones desarrollada enteramente

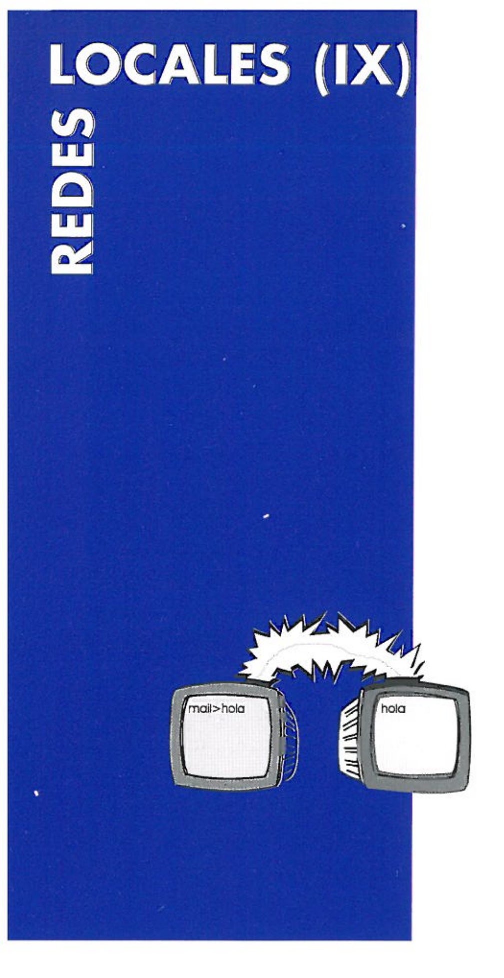
EL EJEMPLO

Se han reescrito muchos de los efectos escritos a lo largo de este curso para ilustrar el manejo de la nueva librería, así como de este nuevo compilador para, de esta forma, poder comprobar el correcto funcionamiento de todas y cada una de las funciones de las librerías MCGA y LIB3DFP.

Otro tipo nuevo que añade la unidad DOS, y que también llama la atención, es el tipo registers

Los efectos incluidos en el código que acompaña a este artículo son las barras de *Copper*, el manejo de sprites, las interferencias, el campo de estrellas, la lente, el rellenado plano y el rellenado *gouraud*.

Se está trabajando en unas páginas web, que se encuentran en la dirección internet http://www.tmt.ergos.es, donde los lectores podrán enviar su código, sugerencias o preguntas al respecto de este curso y sus librerías. Además, se intentará estar al día en lo que a novedades y variaciones del compilador se refiere.



PROTOCOLOS DE COMUNICACIONES: IPX/SPX

María Jesus Recio

Aunque hoy en día los protocolos más populares son TCP/IP, entre otras cosas por su utilización en Internet, los protocolos IPX/SPX siguen teniendo su importancia, fundamentalmente cuando se habla de redes con el sistema operativo Netware de Novell que, aunque puede funcionar con otras pilas de protocolos, fue concebido para trabajar con IPX/SPX, dotando con ellos a la red de ciertas ventajas.

En este artículo se va a hacer un repaso sobre otra familia de protocolos, también muy conocida, como es el caso de los protocolos IPX/SPX, utilizados en redes con el sistema operativo Netware de Novell.

Antes de entrar en detalle sobre cada uno de estos protocolos, es fundamental su ubicación dentro de la pila teórica de protocolos:

- IPX (*Internetwork Packet Exchange* (intercambio de paquetes entre redes)) es un protocolo situado a nivel de red.
- SPX (*Sequenced Packet Exchange* (intercambio secuencial de paquetes)) es un protocolo situado a nivel de transporte, es decir, IPX se encuentra al mismo nivel que el protocolo IP y SPX se encuentra al mismo nivel que TCP. Por tanto, se puede decir de forma muy genérica que deben cubrir las mismas necesidades respectivamente que los protocolos citados.

CONCEPTOS PREVIOS

Antes de iniciar la explicación detallada del esquema de funcionamiento de estos protocolos, es conveniente recordar algunos términos que van a utilizarse en este artículo, y que son necesarios para el claro entendimiento del mismo:

- Paquete: Grupo de bits que contienen datos e información de cabecera, que son enviados por la red como flujos de bits, en tramas, a través de un canal de comunicación a un destino. Un paquete que quiere ser enviado, a veces, es necesario fragmentarlo, introduciendo cada parte en la que se ha dividido en una trama junto con información de control.
- Trama: Estructura que define cómo los datos y la información de control se acomodan al flujo de bits que posibilita la comunicación. Es la forma de llamar a la información transmitida a nivel más bajo. En comunicaciones síncronas cada trama se separa de las demás por un intervalo de tiempo. En comunicaciones asíncronas cada trama incorpora un bit de comienzo y otro de final. La trama transmitida puede ser de longitud fija o de longitud variable.
- Datagrama: Nombre asignado a los paquetes transferidos en una comunicación no orientada a la conexión.
- Comunicación no orientada a la conexión: En este tipo de comunica-

Figura 1

FIGURA 1: Asignación de protocolos Netware para el modelo de referencia OSI.

Modelo OSI	Protocolos Netware				
Aplicación	Aplicaciones Netbios	SAP	NCP	Aplicaciones Netware	
Presentación					
Sesión	Netbios				
Transporte				SPX	
Red	IPX		NLSP		RIP
Enlace	Protocolos MAC (Ethernet, Token-ring)				
Físico					

LISTADO 1
EJEMPLO DE UN FICHERO STARTNET.BAT

```
SET NWLANGUAGE= ESPANOL
C:\NWCLIENT\LSL.COM
C:\NWCLIENT\NE2000.COM
 #driver de la tarjeta de red instalada
C:\NWCLIENT\IPXODI.COM
C:\NWCLIENT\VLM.EXE
```

ción, cada paquete de datos (datagrama) es una unidad independiente, que viaja por sí misma a través de la red. No existe una negociación inicial entre el emisor y el receptor. El emisor únicamente comienza el envío de datagramas sobre la red. Por tanto, cada paquete transferido debe contener la dirección de origen y la dirección de destino. En este tipo de comunicación no existe ningún reconocimiento por parte del receptor sobre los paquetes que recibe, existiendo una ausencia de control de flujo (de controlar que los paquetes podrían llegar fuera de secuencia), con lo que el receptor deberá organizar dicha secuencia. Habitualmente, todos los protocolos no orientados a la conexión residen en el nivel de red del modelo OSI. Un ejemplo de la vida real que podría encontrar una similitud muy elevada con este tipo de comunicación es el servicio de correos: cada carta enviada viaja de manera independiente del resto, incluso de cartas que salgan del mismo origen y tengan el mismo destino. Podría darse la situación de que la última que se mandó fuera la primera en llegar. Cada carta, además, lleva la dirección completa tanto del destinatario como del remitente.

- Comunicación orientada a la conexión: En este tipo de comunicación se establece un canal (circuito) para la comunicación de datos entre los nodos finales. Ese canal es de naturaleza lógica, llamándose por ello a menudo circuito virtual. Este canal se caracteriza por mantener la conexión entre las estaciones finales y no por establecer un trayecto físico a través de la red. De hecho, en redes que presentan múltiples trayectos a un determinado destino, el camino físico puede cambiar durante la sesión, con objeto de acomodarse a las patrones de tráfico establecidos. Debido a que los paquetes se transmiten sobre un circuito virtual, las direcciones completas de los paquetes no son necesarias, puesto que la red conoce las direcciones origen y destino. En realidad, una comunicación de este tipo tiene tres fases: establecimiento de la conexión, intercambio de información y liberación de la conexión. Estos protocolos son utilizados ampliamente en el nivel de transporte del modelo OSI. Un ejemplo de la vida real que sirve para entender más claramente este tipo de comunicación son las comunicaciones telefónicas: cuando se quiere hablar con alguien, primero es necesario marcar su número de teléfono y esperar a que descuelgue (establecimiento de la conexión). A partir de aquí ya se puede establecer el intercambio de información y para terminar es necesario que se cuelgue el teléfono.

ARQUITECTURA DE PROTOCOLOS NETWARE DE NOVELL.

Los protocolos Netware de Novell tiene su origen en la arquitectura Xerox XNS (*Xerox Network System*), desarrollada en los años 70. Estos protocolos están diseñados para sostener la arquitectura cliente-servidor que implementa Netware, por tanto, para optimizar la comunicación entre los servidores y las estaciones cliente.

Para la arquitectura Netware existen tres tipos de máquinas en la red:

- SERVIDORES DE FICHEROS: Controlan las operaciones de la red, así como realizar las funciones de planificación de ficheros, centralización de la información, etc.
- ESTACIONES DE TRABAJO: Son las máquinas de la red que acceden a los servidores para obtener información de éstos. Las estaciones de trabajo funcionan como máquinas independientes, con toda su potencia, hasta que necesitan información localizada en un servidor. Es en este momento cuando realizan la petición a un servidor de la red.
- *ROUTERS*: Se trata de aquellas máquinas que permiten encaminar la información que se transmite ente varios servicios de red. Los *routers* se implementan, la mayoría de las veces, en los servidores de ficheros.

En esta arquitectura de protocolos se definen hasta cinco tipos de servicios básicos:

- Servicio de envío sin conexión.
- Servicio de información de encaminamiento.
- Servicio de información de servidores.
- Servicio de transporte de datos fiable.
- Servicio de interacción cliente-servidor.

PROTOCOLO IPX

Antes de empezar a explicar el protocolo IPX debe señalarse que se va a hablar del esquema de funcionamiento de este protocolo no sólo cuando se trabaja con una red con un solo seg-

TABLA 1 TIPOS DE PAQUETES DE NETWARE

Tipo de paquete	Valor del campo (Hex)	Finalidad
Protocolo de servicio de enlace de Netware.	0x00	Paquete NLSP
Información de encaminamiento	0x01	Paquete RIP
Notificación de servicio	0x04	Paquete SAP
Secuenciado	0x05	Paquete SPX
Protocolo Principal de Netware	0x11	Paquete NCP
Propagado y otros paquetes pro pagados.	0x14	NetBIOS

FIGURA 2: Estructura de un paquete IPX.

Figura 2
Suma de comprobación (2 bytes)
Longitud del paquete (2 bytes)
Control de transporte (1 byte)
Tipo de paquete (1 byte)
Red de destino (4 bytes)
Nodo de destino (6 bytes)
Socket de destino (2 bytes)
Red de origen (4 bytes)
Nodo de origen (6 bytes)
Socket de origen (2 bytes)
datos

mento, sino cuando se tiene una red con varios segmentos y también cuando se interconectan varias redes. Debido a esto aparecen conceptos como encaminamiento, *routers*, etc..., que no tendrían sentido si el artículo se centrara únicamente en una red sencilla, sin segmentar.

Se trata, como ya se ha dicho, de un protocolo de datagramas que opera en el nivel de red del modelo de protocolos OSI (ver figura 1). Es, por tanto, un protocolo no orientado a la conexión, por lo que todos los paquetes (datagramas) IPX que contienen datos se direccionan y se envían a sus destinos, pero no se garantiza ni verifica que el paquete haya llegado correctamente, puesto que cualquier reconocimiento de paquete o control de conexión es proporcionado por los protocolos que aparecen por encima de IPX, tal como el protocolo SPX.

LISTADO 2
EJEMPLO DE UN FICHERO NET.CFG

```
Ejemplo de un fichero net.cfg

Link Driver NE2000
PORT 300
IRQ 5
FRAME Ethernet 802.2

Netware DOS Requester
FIRST NETWARE DRIVE = F
NETWARE PROTOCOL = NDS BIND
NAME CONTEXT =
"despacho.habitacion.coco"
```

Por trabajar a nivel de red, IPX se encarga de direccionar y encaminar paquetes desde una ubicación a otra de una inter-red IPX. IPX basa sus decisiones de encaminamiento en los campos de dirección de cabecera y en la información que recibe desde los protocolos RIP y NLSP.

DIRECCIONAMIENTO IPX

Una dirección IPX tiene 3 partes, cada una de las cuales aparece en un campo diferente del paquete IPX. Estas partes son:

- Dirección de red.
- Dirección de nodo.
- Dirección de *socket*.

La dirección de red es un número hexadecimal de cuatro *bytes* que sirve de base para el encaminamiento entre redes o segmentos IPX. A cada segmento de una red, así como a cada red, se le asocia una dirección de red diferente. De esta forma, cuando un paquete IPX llega a un *router*, éste, examinando la dirección de red, sabe en qué segmento está el destino y puede encaminar dicho paquete hacia él.

Esta dirección de red puede contener hasta 8 dígitos incluyendo los ceros. Por ejemplo, FEDCBA98, 1234567D y C7 son direcciones de red válidas. Las direcciones 0 y FFFFFFFF no se encuentran disponibles para el direccionamiento de red, ya que son utilizadas por los servidores para propósitos especiales.

La dirección de nodo es un número hexadecimal de seis *bytes* que identifica un dispositivo (servidor, estación cliente, *router*, etc.) en una red o segmento IPX. La dirección de nodo coincide con la dirección física de la tarjeta de red instalada en el dispositivo.

La dirección de *socket* es un número hexadecimal de dos *bytes* que identifica un proceso dentro de una máquina de la red (un nodo). Debido a que en un nodo pueden coexistir de manera simultánea varios procesos, esta dirección identifica el proceso al que el paquete va dirigido. Cuando un proceso quiere utilizar la red, (hacer una petición a un servidor, por ejemplo) es necesario que se le asigne un número de *socket*. En la tabla 2 se detallan algunos números de *sockets* y procesos que se utilizan en el entorno Netware.

Los números de *socket* entre 0x4000 y 0x7FFF son *sockets* dinámicos, utilizados por las estaciones de trabajo para comunicar con servidores de ficheros y otros dispositivos de la red.

El lector debería haber encontrado cierta similitud entre el direccionamiento IPX y el ya conocido direccionamiento utilizado por los protocolos TCP/IP. Recuérdese que la dirección IP estaba dividida en dos partes: una dirección de red, una dirección de máquina dentro de la red y que TCP añadía una dirección de puerto que coincida con el servicio solicitado/prestado.

ESTRUCTURA DEL PAQUETE IPX

Como todos los paquetes, los paquetes IPX constan de dos partes:

- Cabecera IPX: Su tamaño es de 30 bytes e incluye las direcciones de red, nodo y *sockets* tanto para el destino como para el origen.
- Datos: Es la parte en la que se encapsula el paquete que viene de nivel superior y que, por tanto, tendrá información constituida en último extremo por la información que se quiere transmitir propiamente dicha, así como la cabecera de control añadida por el protocolo de nivel superior (en este caso SPX). La longitud de esta parte es variable.



TABLA 2 PROCESOS Y NÚMEROS DE SOCKETS DE NETWARE

Número de socket	Proceso
0x451	Protocolo principal de Netware (NCP)
0x452	Protocolo de notificación de servicios (SAP)
0x453	Protocolo de información de encaminamiento (RIP)
0x455	NetBIOS de Novell
0x456	Diagnósticos
0x9001	Protocolos de servicios de enlace de Netware (NLSP)
0x9004	Protocolo IPXWAN

El tamaño mínimo de un paquete IPX es de 30 *bytes*, es decir, solamente la cabecera IPX sin ninguna información. El tamaño máximo antiguamente era de 576 bytes, es decir, 546 bytes íntegros de información, aunque en la actualidad se aceptan paquetes de hasta 65.535 bytes en total, siendo las limitaciones del medio físico las que determinan el tamaño del paquete.

La estructura de un paquete IPX aparece reflejada en la figura 2. De todas las partes de un paquete IPX cabe destacar las siguientes:

- Tipo de paquete: Indica el servicio ofrecido o requerido por el paquete. Los tipos de paquetes pueden ser los que aparecen en la tabla 1.
- Dirección de red origen/destino: Se trata de la dirección de red que tiene asignada la estación que solicita la comunicación y la dirección de red que tiene asignada la estación a la que se le solicita la comunicación, respectivamente.
- Dirección de nodo origen/destino: Se trata de la dirección física de la tarjeta de red implantada en la estación origen y en la estación destino.
- Dirección de *socket* origen/destino: Es un identificador del proceso que realiza la petición y del proceso en la máquina destino al que se realiza dicha petición.

PROTOCOLO SPX

Como ya se ha dicho, IPX ofrece un servicio de envío de paquetes sin conexión, por lo que no garantiza la entrega de dichos paquetes ni su duplicación ni, por supuesto, el orden en el que son entregados.

IPX, por tanto, necesita de un servicio que le proporcione una gestión de conexión entre procesos remotos, y para ello ha sido diseñado el protocolo SPX (*Sequenced Packet Exchange*) localizado, como ya se mencionó, en el nivel de transporte del modelo OSI.

SPX, en una red Netware, es el protocolo que se encarga de garantizar el envío y de mantener el orden en el flujo de los paquetes que componen un mensaje, para lo cual establece una conexión entre las estaciones que participan en la comunicación (se trata, por tanto, de un protocolo orientado a la conexión).

Mientras que los protocolos no orientados a la conexión son más eficientes cuando no existe una comunicación continua entre dos máquinas, los protocolos orientados a conexión son mejores cuando va a haber una conexión relativamente permanente, como por ejemplo si se ejecuta la utilidad consola remota de Netware. En este, caso como la conexión va a ser continua, es mejor establecerla previamente.

Un paquete SPX tiene una cabecera de 13 bytes divididos como se muestra en la tabla 3.

FUNCIONAMIENTO DE NETWARE DE NOVELL

Al instalar una red Netware, es decir, una o más estaciones servidor y una o más estaciones cliente, se obtiene el medio de acceder a los servicios prestados por las estaciones servidor.

TABLA 3 CABECERA DE UN PAQUETE SPX
Control de conexión (1 bytes)
Tipo de flujo de datos (2 bytes)
Identificador de la conexión de origen (2 bytes)
Identificador de la conexión de destino (2 bytes)
Número de secuencia (2 bytes)
Número de reconocimiento (2 bytes)
Número de asignación (2 bytes)

Cuando se realiza una petición en una estación cliente, un programa (un redirector, que en el caso de Netware es el *DOS Requester*), examina dicha petición: si se trata de una petición dirigida a la propia máquina (petición local) se la pasa al sistema para que éste la resuelva. En el caso de tratarse de una petición dirigida al servidor, realiza las operaciones oportunas para que llegue a su destino dicha petición.

Para facilitar la utilización entre estaciones cliente y servidores de recursos es necesario que, además de unos protocolos que proveen acceso a la red e información sobre direcciones y servicios, exista un mecanismo que convierta las órdenes de las aplicaciones de usuario en peticiones y respuesta a servidores.

FICHEROS DE CONFIGURACIÓN DE UN CLIENTE NETWARE

Existen dos ficheros de configuración básicos en una estación cliente de Netware que son: startnet.bat y net.cfg.

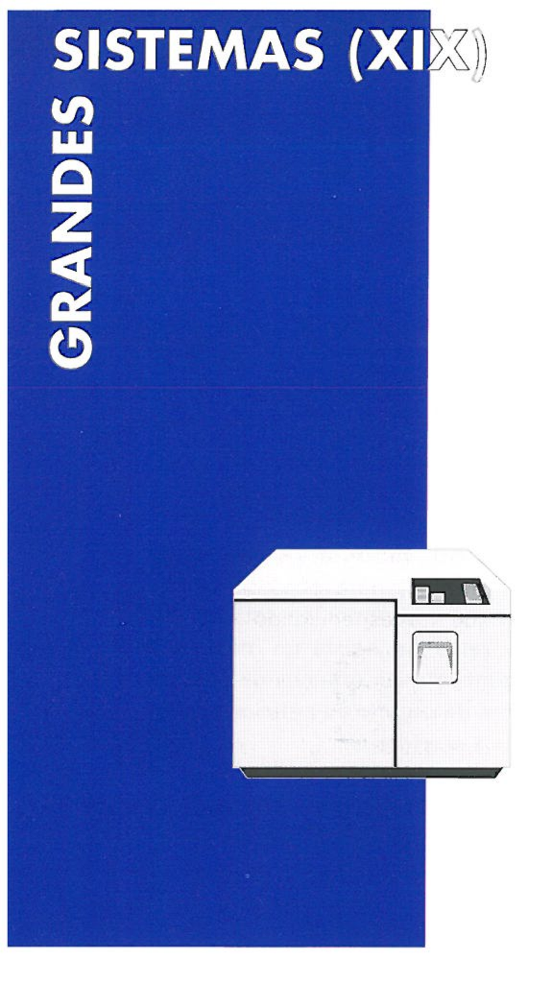
El primero de ellos, el fichero startnet.bat, no es un fichero de configuración propiamente dicho. Se trata más bien del fichero que contiene los módulos que se deben cargar en una estación cliente para que ésta funcione correctamente. El listado 1 muestra un ejemplo típico de este fichero para la versión 4.1 de Netware de Novell.

El segundo fichero citado, net.cfg, sí es un fichero de configuración. En él se definen, entre otras cosas, el tipo de trama que se va a utilizar, la primera letra asignada a un disco del servidor, la IRQ y la dirección que Netware utiliza en esta máquina, el nombre del contexto, etc...

Es importante destacar que, para un correcto funcionamiento de una estación cliente, la IRQ y la dirección dadas en este fichero deben coincidir con las que tiene en su configuración la tarjeta de red. El listado 2 muestra un ejemplo de un fichero net.cfg.

PRÓXIMO NÚMERO

En la próxima entrega se seguirá hablando de Netware de Novell más en profundidad. Se verá con más detalles el fichero net.cfg, otros protocolos involucrados y la instalación tanto de un servidor de ficheros de Novell como de una estación cliente.



GESTIÓN DE ARRAYS EN EL ENTORNO NATURAL

José María Peco

NATURAL realiza la gestión de tablas o arrays en los programas de una forma ágil y cómoda, y lo mismo ocurre con la información que se puede obtener de las variables de tipo fecha (*DATE*) tal y como se muestra en el ejemplo que acompaña al artículo.

El hecho de trabajar en grandes sistemas con terminales 3270, que son de sólo texto, hace difícil mostrar todo un año con el formato de "calendario tradicional" en una única pantalla. Por esta razón sólo presentarán semestres por pantalla, mientras que en el listado, se sacará el año completo.

NATURAL realiza la gestión de tablas en los programas de una forma cómoda

El disponer de un ejemplo completo y claro favorece que el programador utilice mejor todos los recursos que pone a su disposición el sistema. Tal es el objetivo buscado con el presente artículo, el cual, tal y como indica su título, presenta un ejemplo muy completo sobre el tratamiento de las tablas o arrays, así como del tratamiento de fechas en el entorno NATURAL.

TABLAS USADAS

La **figura 1** muestra el calendario del año 1997; mientras que la **figura 2** ayuda a identificar las 5 tablas que se usan en el mapa {**CALENDAM**} y que se comentan a continuación:

T-DIA-MES (N2/1:12,1:6,1:7) Tabla para contener cada día de cada uno de los meses: Numérico de 2 dígitos, y tres dimensiones (12 meses; 6 semanas por mes/7 días por semana)

T-LIT-MES (A12/1:12) : Tabla para contener los nombres de los doce meses.

Para contener la cabecera del día de la semana, se recurre a una tabla redefinida. (ver líneas 350-400 del listado de **CALENDAP**). Realmente lo que se hace es reservar un espacio de 42 bytes de memoria (*T-LIT-DIA-SEM*). Ese espacio, se redefine como una tabla de 7 posiciones y 6 ocurrencias, que sólo sirve para facilitar la carga de los literales (*T-LIT-DIA*). Y, por último, ese mismo espacio de 42 posiciones, se

Figura 1

```
Calendario correspondiente al año: 1997 o MCMXCVII          comienza en MIERCOLES

        Enero                      Febrero                    Marzo

    L  M  X  J  V  S  D        L  M  X  J  V  S  D        L  M  X  J  V  S  D
 1        1  2  3  4  5     5                 1  2     9                 1  2
 2  6  7  8  9 10 11 12     6  3  4  5  6  7  8  9    10  3  4  5  6  7  8  9
 3 13 14 15 16 17 18 19     7 10 11 12 13 14 15 16    11 10 11 12 13 14 15 16
 4 20 21 22 23 24 25 26     8 17 18 19 20 21 22 23    12 17 18 19 20 21 22 23
 5 27 28 29 30 31           9 24 25 26 27 28          13 24 25 26 27 28 29 30
                                                      14 31

        Abril                      Mayo                       Junio

    L  M  X  J  V  S  D        L  M  X  J  V  S  D        L  M  X  J  V  S  D
14     1  2  3  4  5  6    18           1  2  3  4    22                    1
15  7  8  9 10 11 12 13    19  5  6  7  8  9 10 11    23  2  3  4  5  6  7  8
16 14 15 16 17 18 19 20    20 12 13 14 15 16 17 18    24  9 10 11 12 13 14 15
17 21 22 23 24 25 26 27    21 19 20 21 22 23 24 25    25 16 17 18 19 20 21 22
18 28 29 30                22 26 27 28 29 30 31       26 23 24 25 26 27 28 29
                                                      27 30
```

Figura 1: Resultado perseguido : Calendario del año 1997

redefine como una tabla de 3 dimensiones (*T-DIA-L*: 2 filas * 3 meses * 7 días por mes):

T-NUM-SEMANA (n2/1:12,1:6): Esta tabla contendrá el número de semana (divida en 12 meses y hasta 6 semanas por mes)

T-DIA-CV (c/1:12,1:6,1:7): Esta tabla se define para contener variables de control, a fin de que los días festivos aparezcan con más brillo (intensificados).

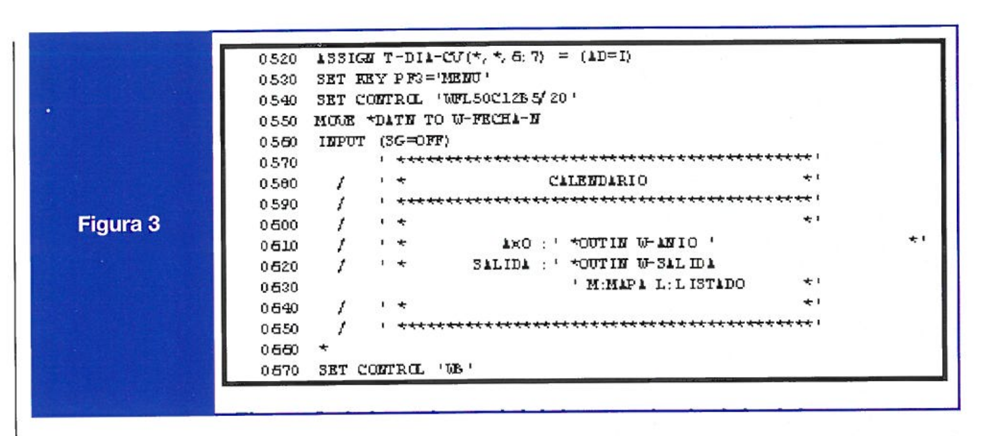
El resto de variables pueden verse en los listados de los fuentes que se acompañan, tanto de **CALENDAM** y **CALENDAP**

COMENTARIOS AL PROGRAMA

La **figura 3** muestra la cabecera del programa, donde cada línea tiene el siguiente significado:

- **0520**: Carga todas las ocurrencias 6 y 7 de todos los meses y de todas las semanas de la tabla de variables de control con el atributo de intensificado.
- **0530**: Activa la tecla PF3 y la asigna el comando "MENU"
- **0540**: Crea una ventana de 50*12, y sitúa el ángulo superior izquierdo en la línea 5 columna 20.
- **0550**: Carga la variable W-FECHA-N con el valor numérico de la fecha actual, para que en la ventana ya salga cargado el dato del año a listar con el valor del año actual.
- **0560-0650**: Petición al usuario de año que se desea listar, depositando este valor en la variable W-ANIO, que a su vez es un subcampo de W-FECHA-N. El parámetro *OUTIN que precede a la variable es para especificar que dicho campo es de salida (OUT) y entrada (IN), ya que en caso de no especificarle asume que es *IN (sólo entrada)
- **0620**: Dentro del mismo input se pide al usuario que elija si desea la salida mediante mapa o mediante listado realizado con write.
- **0670**: Restaura los atributos de pantalla a los valores por defecto. Es decir cierra la ventana.

Figura 3



```
0520  ASSIGN T-DIA-CV(*,*,6:7) = (AD=I)
0530  SET KEY PF3='MENU'
0540  SET CONTROL 'WFL50C12B5/20'
0550  MOVE *DATN TO W-FECHA-N
0560  INPUT (SG=OFF)
0570          '*****************************************'
0580    /     '*                CALENDARIO             *'
0590    /     '*****************************************'
0600    /     '*                                       *'
0610    /     '*          AÑO :' *OUTIN W-ANIO '                *'
0620    /     '*       SALIDA :' *OUTIN W-SALIDA
0630                          ' M:MAPA L:LISTADO      *'
0640    /     '*                                       *'
0650    /     '*****************************************'
0660  *
0670  SET CONTROL 'WB'
```

La **figura 4** muestra un fragmento de programa correspondiente a la *elaboración de datos*, según el esquema propuesto en los artículos de esta sección (Sólo Programadores num. 24 y 25)

Para contener la cabecera del día de la semana se recurre a una tabla redefinida

- **0750**: Establece como lenguaje del sistema el Castellano, para que los valores devueltos por el sistema vengan en este lenguaje.

Figura 2

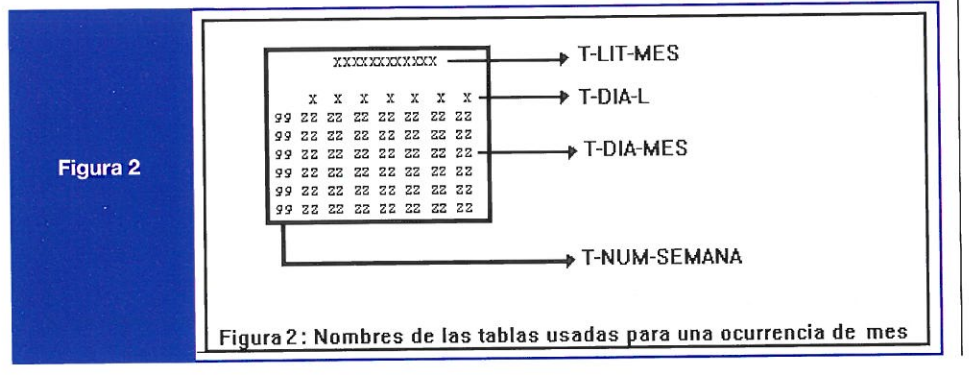


Figura 2: Nombres de las tablas usadas para una ocurrencia de mes

- **0760**: Inicializa los valores de día y mes a 1 para obtener, junto al año que ha especificado el usuario, el primer día del año.
- **0790**: Mueve el contenido del campo numérico que contiene la fecha (primer día del año), a una variable de tipo '*date*' utilizando la sentencia MOVE EDITED. Esta sentencia, por regla general, cuando se usa con variables de tipo *Date(D)*, usa como variable receptora o emisora una variable de tipo alfabética, de ahí que se utilice la variable W-FECHA-A que es la redefinición de W-FECHA-N (numérica).
- **0800**: Obtiene el año romano de la fecha.
- **0810**: Obtiene el día de la semana de la fecha contenida en la variable de tipo date.
- **0830**: Pasa a mayúsculas el día de la semana obtenido en 820, ya que el sistema deja este valor con el primer carácter en mayúsculas y el resto en minúsculas, y para la siguiente validación se necesita que todo este en mayúsculas, a menos que cuando se está editando el programa, se haya especificado el comando de terminal **%L** para que no se pasen a mayúsculas los valores que se escriban en las líneas 850-910.
- **0840-0930**: Carga la variable auxiliar w-aux con el número del día de la semana.

La siguientes instrucciones no tienen mayor complejidad, por lo que pueden seguirse en el listado del programa que se acompaña en el disco.

- **0950-0990**: se corresponden con la carga de los nombres de los días de la semana moviendo el literal 'LMXJVSD'

Figura 4

```
0750   MOVE  4 TO *LANGUAGE
0760   MOVE 01 TO W-MES
0770   MOVE 01 TO W-DIA
0780   *
0790   MOVE EDITED W-FECHA-A                  TO W-DATE (EM=YYYYMMDD)
0800   MOVE EDITED W-DATE     (EM=R)          TO W-ANIO-ROMANO
0810   MOVE EDITED W-DATE     (EM=N(10))      TO W-LIT-DIA1
0820   *
0830   EXAMINE W-LIT-DIA1 TRANSLATE INTO UPPER CASE
0840   DECIDE ON FIRST VALUE OF W-LIT-DIA1
0850       VALUE 'LUNES'      MOVE 1 TO W-AUX
0860       VALUE 'MARTES'     MOVE 2 TO W-AUX
0870       VALUE 'MIERCOLES'  MOVE 3 TO W-AUX
0880       VALUE 'JUEVES'     MOVE 4 TO W-AUX
0890       VALUE 'VIERNES'    MOVE 5 TO W-AUX
0900       VALUE 'SABADO'     MOVE 6 TO W-AUX
0910       VALUE 'DOMINGO'    MOVE 7 TO W-AUX
0920       NONE VALUE IGNORE
0930   END-DECIDE
0940   *
```

Figura 4: Tratamiento de fechas

- **1030-1050**: Estructura repetitiva para tratar mediante FOR cada uno de los 12 meses de un año.

La **figura 5** muestra, quizás, la parte más complicada del programa, pues carga los datos variables correspondientes a un mes.

Los índices que se usan *para las distintas repetitivas tipo FOR, son los subcampos W-MES y W-*DIA redefinidos de la variable *W-FECHA-N.*

La variable auxiliar *W-AUX* se utiliza para ir arrastrando el día dentro de la semana de un mes a otro, por eso, cada vez que se carga un día, se suma 1 a esa variable, y cuando supera el valor 7 actualiza los demás índices.

- **1190 - 1200**: Obtiene el nombre del mes desde el sistema.
- **1210-1220**: Obtiene el numero de la semana del primer día del mes.
- **1270-1460**: repetitiva tipo FOR para cargar los 31 días del mes.

COMENTARIOS AL MAPA

Los comentarios asociados al mapa, se tienen que realizar sobre los hardcopys que muestran el interface proporcianado por NATURAL para el tratamiento de los mapas, ya que este entorno no admite otra forma de definición de mapas, a no ser por mapping directo en el propio programa.

La **figura 6** muestra las variables usadas en el mapa **CALENDAM**. Es de resaltar que las tablas T-NUM-SEMANA y T-DIA-MES se encuentran repetidas, pues se tratan por trimestre.

La **figura 7** muestra el detalle correspondiente a la tabla T-LIT-MES. Esta tabla, de 1 dimensión con 12 ocurrencias, es tratada como una tabla de 3*2. Para lo cual, a la izquierda de *INDEX HORIZONTAL* se pone un 1 por ser la primera dimensión y querer que las ocurrencias aparezcan en horizontal, pero solo tres ocurrencias, y además comenzando por el valor contenido en la variables *W-IND-1.* Al colocar un 2 en *INDEX VERTICAL* hace que se listen las siguientes ocurrencias como si fueran las dos dimensiones citadas. En resumen, solo se listan las 6 ocurrencias que se necesitan.

La **figura 8** muestra el detalle de la definición de la tabla *T-NUM-SEMANA.* Esta tabla, de 2 dimensiones (12*6), es tratada como si fuera de 3*6, donde el primer índice establece el trimestre, pues presenta 3 ocurrencias comenzando por la especificada en la variable *W-IND-1*, y el segundo, el número de semana dentro del mes.

La **figura 9** muestra la definición artículo compleja ya que hace uso de las tres dimensiones para representar la tabla *T-DIA-L.* Esta tabla, como puede verse en la primera línea, es una tabla de 3 dimensiones, concretamente de 2*3*7 ocurrencias. Donde:

- la primera dimensión, (con un "1" bajo el titulo "*Dimensions*", con 2 ocurrencias, especifica el trimestre (por decirlo de alguna forma) ya que se corresponde con el índice vertical (Espaciado entre ocurrencias: 10 líneas).
- la segunda dimensión, con 3 ocurrencias, corresponde con cada uno de los meses dentro del trimestre (espaciado entre ocurrencias: 7 columnas según se aprecia en la figura); y, por último,
- la tercera dimensión establece las 7 ocurrencias de cada mes, escritas en sentido horizontal, por eso la opción (**H**) en el campo *"INDEX"*. dejando entre ocurrencia y ocurrencia 2 columnas.

La **figura 10** muestra la definición de la tabla que se corresponde con los días de cada mes, siendo tratada ("*pinchada*") por trimestre, por eso se

Figura 5

```
1140   ***********************************************************
1150   DEFINE SUBROUTINE CARGAR-MES
1160   *
1170   MOVE 1 TO W-SEMANA
1180   MOVE # TO W-DIA
1190   MOVE EDITED W-FECHA-A              TO W-DATE (EM=YYYYMMDD)
1200   MOVE EDITED W-DATE (EM=L(12))      TO T-LIT-MES (W-MES)
1210   MOVE EDITED W-DATE (EM=ZW)         TO W-NUM-SEMANA-A
1220   MOVE W-NUM-SEMANA-N                TO T-NUM-SEMANA (W-MES, 1)
1230   IF W-MES = 1
1240      MOVE 0 TO W-NUM-SEMANA-N
1250   END-IF
1260   *
1270   FOR W-DIA = 1 TO 31
1280       IF W-FECHA-N NE MASK (YYYYMMDD)
1290          ESCAPE BOTTOM
1300       END-IF
1380       MOVE W-DIA TO T-DIA-MES (W-MES, W-SEMANA, W-AUX)
1390       ADD 1 TO W-AUX
1400       IF W-AUX GT 7
1410          MOVE 1 TO W-AUX
1420          ADD  1 TO W-SEMANA
1430          ADD  1 TO W-NUM-SEMANA-N
1440          MOVE W-NUM-SEMANA-N  TO T-NUM-SEMANA (W-MES, W-SEMANA)
1450       END-IF
1460   END-FOR
1470   *
1480   IF W-AUX = 1
1490      MOVE 0 TO T-NUM-SEMANA (W-MES, W-SEMANA)
1500   END-IF
1510   *
1520   END-SUBROUTINE
```

Figura 5: Tratamiento de un mes.

Figura 6

```
17:00:55          Field and Variable Definitions - Summary           96-10-30

Cmd Field Name (Truncated)                            Mod Format Ar Ru Lin Col
___ W-ANIO___________________________________________ U   N4              1  36
___ W-ANIO-ROMANO____________________________________ D   A13             1  43
___ W-LIT-DIA1_______________________________________ D   A10             1  70
___ T-LIT-MES________________________________________ D   A12    A        3   9
___ T-LIT-DIA-SEM.T-DIA-L____________________________ D   A1     A        5   7
___ T-NUM-SEMANA______   <==== Repetido______________ D   N2     A        6   3
___ T-DIA-MES_________   <==== Repetido______________ D   N2     A        6   6
___ W-IND-1__________________________________________ D   N3             12  37
___ W-IND-2__________________________________________ D   N3             12  42
___ T-NUM-SEMANA______   <==== Repetido______________ D   N2     A       17   3
___ T-DIA-MES_________   <==== Repetido______________ D   N2     A       17   6



Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Mset  Exit              --                Parm  Local       Let
```

Figura 6: Variables usadas en el mapa CALENDAM

encuentra repetida su definición en la listado de variables del mapa (**figura 6**). Concretamente, la figura muestra la definición correspondiente al segundo trimestre del semestre que se muestre, iniciándose éste por el mes especificado en la variable W-IND-2. La primera dimensión establece el trimestre. La segunda, establece la semana (en sentido vertical), y la tercera, se escribe en horizontal, por eso el valor (**H**) en el campo *INDEX*, y tiene 7 ocurrencias.

Por último, la **figura 11** muestra la definición de la variable W-IND-1, utilizado como índice de alguna tabla del mapa. Este campo realmente solo sirve para que aparezca definido en el área de parámetros del objeto mapa (**CALENDAM**), ya que en el caso de no hacerlo así, se produciría un error de parámetros al ser invocado desde el programa. Asi pues, como no tiene otra función, se le asigna el atributo (**AD=N**) de no visualizable (Non-Display), y una longitud de numérico (**NL=1**) de 1 dígito, a pesar de estar definida como (**N03**) numérico de 3 dígitos.

SENTENCIAS ASOCIADAS CON EL TRATAMIENTO DE TABLAS

NATURAL también dispone de otras sentencias para la definición y gestión de las tablas. Este apartado, pretende recordar aquellas instrucciones que sin estar incluidas en el ejemplo, están asociadas con el tema tratado.

Así, en el caso de querer saber cual es la suma de todas las ocurrencias de los días de un mes, sólo habría que hacer:

W-Suma = + t-dia-mes (1,,*)*

Otra situación muy normal en cualquier aplicación consiste en seleccionar una ocurrencia, por ejemplo una provincia, de entre las que encuentran en una lista. Para ello, se suele utilizar otra tabla auxiliar para que el usuario marque en ella, la provincia elegida.

Las sentencias *EXAMINE* y *IF SELECTION* evitan el tener que generar una estructura repetitiva, tipo *FOR* por ejemplo, para tratar esa tabla auxiliar y saber si se ha seleccionado más de una ocurrencia. Ejemplos:

IF SELECTION NOT UNIQUE
REINPUT '==> error...'
EXAMINE T-MARCA (*) FOR 'x' GIVING INDEX W-INDICE

CASO PARTICULAR

NATURAL permite pasar datos de un programa a otro vía *STACK*, siendo recogidos estos valores en el programa llamado mediante la sentencia **INPUT**. (Sólo Programadores Julio-Agos-Sept-96).

Pues bien, cuando la variable receptora es mayor de 80 posiciones, el sistema devuelve un error diciendo que dicho valor no cabe en una línea. Este error se solventa utilizando en el INPUT la redefinición en forma de tabla de una variable de longitud inferior, tal y como se muestra a continuación.

DEFINE DATA LOCAL
01 W-ENTRADA (A150)
01 REDEFINE W-ENTRADA
02 W-CAMPO (A50/1:3)
...
INPUT W-CAMPO (1:3)

SENTENCIAS ASOCIADAS CON EL TRATAMIENTO DE FECHAS

Por otra parte, y con el ánimo de no dejar tampoco en el tintero las otras sentencias de NATURAL para el tratamientos de fechas, se incluye este apartado.

SETTIME

Esta sentencia establece el inicio de un contador del sistema (***TIMD**) para medir el tiempo transcurrido desde que se ejecutó dicha sentencia. Pueden establecerse distintos contadores, siempre y cuando se tenga la precaución de etiquetar cada sentencia *SETTIME*.

DEFINE DATA LOCAL

NATURAL también dispone de otras sentencias para la definición y gestión de las tablas

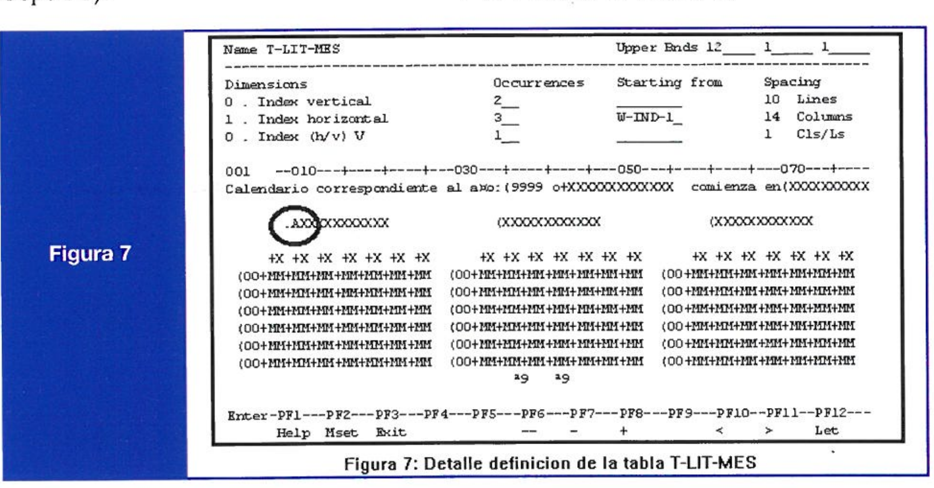


Figura 7

```
Name T-LIT-MES                                    Upper Bnds 12____ 1_____ 1_____
---------------------------------------------------------------------------------
Dimensions                          Occurrences   Starting from    Spacing
0 . Index vertical                  2__           ________         10  Lines
1 . Index horizontal                3__           W-IND-1_         14  Columns
0 . Index (h/v) V                   1__           ________         1   Cls/Ls

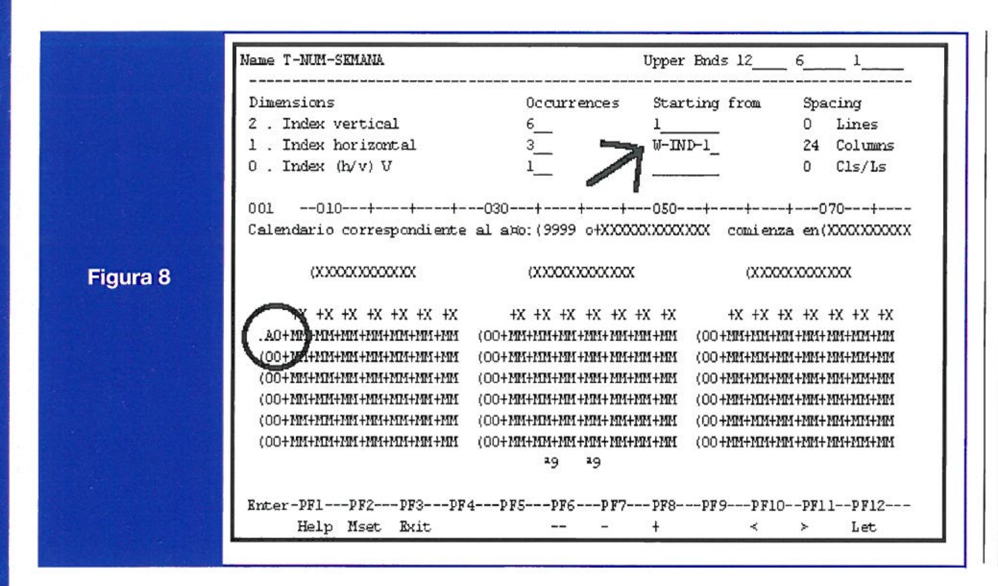
001   --010---+----+----+---030---+----+----+---050---+----+----+---070---+----
Calendario correspondiente al a%o:(9999 o+XXXXXXXXXXXXX  comienza en(XXXXXXXXXX

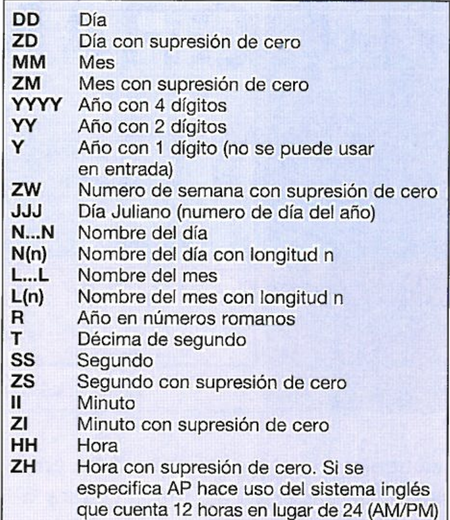
      .AXXXXXXXXXXX                (XXXXXXXXXXXX               (XXXXXXXXXXXX

      +X +X +X +X +X +X +X          +X +X +X +X +X +X +X          +X +X +X +X +X +X +X
   (00+MM+MM+MM+MM+MM+MM+MM   (00+MM+MM+MM+MM+MM+MM+MM   (00+MM+MM+MM+MM+MM+MM+MM
   (00+MM+MM+MM+MM+MM+MM+MM   (00+MM+MM+MM+MM+MM+MM+MM   (00+MM+MM+MM+MM+MM+MM+MM
   (00+MM+MM+MM+MM+MM+MM+MM   (00+MM+MM+MM+MM+MM+MM+MM   (00+MM+MM+MM+MM+MM+MM+MM
   (00+MM+MM+MM+MM+MM+MM+MM   (00+MM+MM+MM+MM+MM+MM+MM   (00+MM+MM+MM+MM+MM+MM+MM
   (00+MM+MM+MM+MM+MM+MM+MM   (00+MM+MM+MM+MM+MM+MM+MM   (00+MM+MM+MM+MM+MM+MM+MM
   (00+MM+MM+MM+MM+MM+MM+MM   (00+MM+MM+MM+MM+MM+MM+MM   (00+MM+MM+MM+MM+MM+MM+MM
                                      a9   a9

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Mset  Exit              --    -     +           <     >     Let
```

Figura 7: Detalle definicion de la tabla T-LIT-MES





DD	Día
ZD	Día con supresión de cero
MM	Mes
ZM	Mes con supresión de cero
YYYY	Año con 4 dígitos
YY	Año con 2 dígitos
Y	Año con 1 dígito (no se puede usar en entrada)
ZW	Numero de semana con supresión de cero
JJJ	Día Juliano (numero de día del año)
N...N	Nombre del día
N(n)	Nombre del día con longitud n
L...L	Nombre del mes
L(n)	Nombre del mes con longitud n
R	Año en números romanos
T	Décima de segundo
SS	Segundo
ZS	Segundo con supresión de cero
II	Minuto
ZI	Minuto con supresión de cero
HH	Hora
ZH	Hora con supresión de cero. Si se especifica AP hace uso del sistema inglés que cuenta 12 horas en lugar de 24 (AM/PM)

```
1 X (N4)
END-DEFINE
S1. SETTIME
WRITE 'HORA INICIO:' *TIME
FOR X = 1 TO 1000
   WRITE X
END-FOR
WRITE 'HORA FIN:' *TIME
/ 'HA TARDADO:' *TIMD(S1.)
END
```

Otro elemento del lenguaje NATURAL muy útil en una instalación es el uso de las máscaras de edición. La siguiente relación muestra aquellas máscaras usadas en combinación con variables de formato **D(date)** o **T(time)**:

Ejemplo:

```
WRITE 'DÍA JUALIANO:' *DATX
(EM=YYJJJ)
  / 'O TAMBIEN:' *DATJ
      / 'HORA INTERNA:' *TIMX
(EM=HH.II.SS.T'-'AP)
```

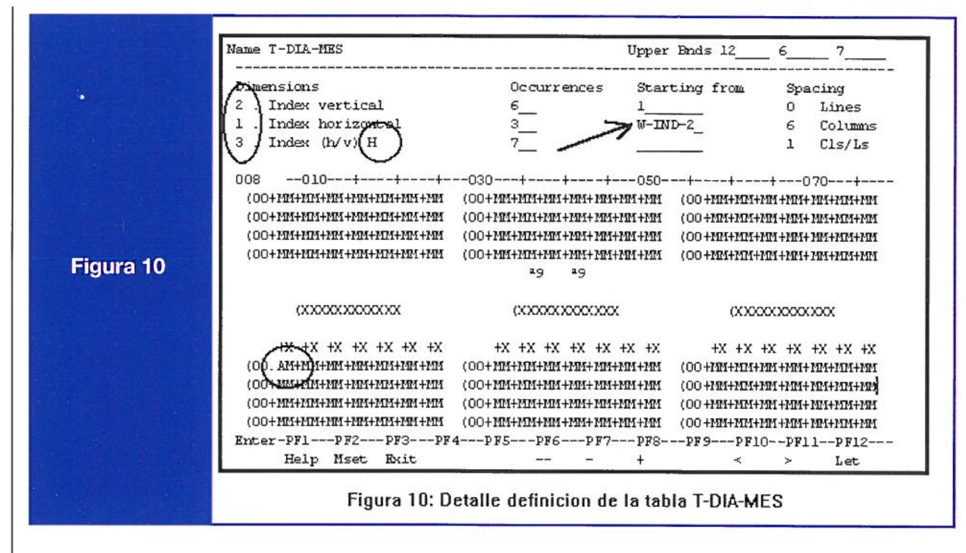


Figura 10

Figura 10: Detalle definicion de la tabla T-DIA-MES

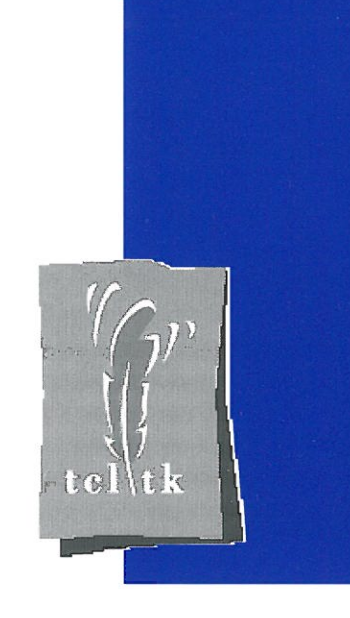
Figura 9

UTILIDAD

Este mes, la utilidad que se acompaña, es la que ha servido de base para el artículo, y con la cual el autor desea felicitar el nuevo año a todos los lectores de Sólo Programadores. No obstante, y con el objeto de no perder la costumbre de acompañar un ejemplo asociado con el tema, en el fichero **JMP_TAB.NAT** se incluye otro programa NATURAL que muestra una forma distinta de tratar los campos redefinidos de una tabla, así como el resultado de su ejecución. Por último, desde estas líneas se lanza a los lectores una invitación para publicar las utilidades desarrolladas por ellos mismos en NATURAL. A tal efecto, deben hacer llegar las mismas, bien a la redacción de la revista, o a la dirección del autor del artículo <jmpeco@luznet.es>. Así mismo, también se admiten direcciones de foros y WEBs relacionadas con el entorno del lenguaje NATURAL.

PROGRAMACIÓN DE INTERFACES GRÁFICAS

Carlos Gerardo Pérez



A partir de este número se inicia una nueva serie de artículos dedicados a la programación de interfaces gráficas con el lenguaje Tcl/Tk. En ellos se tratarán los puntos más relevantes de este lenguaje, partiendo desde sus estructuras lógicas hasta la programación en red.

Para aquellos lectores que estén indecisos sobre si seguir o no esta sección que comienza este mes, decir que en sucesivas entregas se abordarán los siguientes temas:

- La programación con el lenguaje Tcl.
- La creación de Interfaces gráficos con Tk.
- La creación de aplicaciones Tcl en C.
- La creación de nuevas clases de "*Widgets*" usando C y Tk.
- Las comunicaciones con Tcl/Tk.
- La creación de "*tclets*", equivalentes a los "*applets*" de Java, pero programados en Tcl/Tk.

HISTORIA DEL LENGUAJE

El lenguaje Tcl y el "*toolkit*" Tk fueron creados por John K. Ousterhout y su equipo de trabajo en la Universidad de California en Berkeley durante la década de los 80, con el objetivo de tener un lenguaje de comandos que fuesen reutilizables y que facilitara la creación de interfaces gráficas en el entorno X-Windows de UNIX. Sin embargo, lo que en un principio estaba pensado que fuera un lenguaje para la implementación rápida de prototipos se ha convertido en uno de los lenguajes más utilizados en la actualidad en el ámbito académico y con desarrollos que sobrepasan las cien mil líneas de código.

Con el paso de los años, lo que nació como un lenguaje para el sistema operativo UNIX ha sido transportado a otras plataformas. Así, se pueden encontrar versiones para los sistemas operativos OS2, DOS, Machintosh, Windows 95 y Windows NT, aunque estas dos últimas son las únicas versiones que se han mantenido actualizadas, junto con la versión de UNIX, a las versiones finales Tcl/Tk (7.6 y 4.2) respectivamente, que salieron el pasado mes de Octubre. Las últimas versiones se incluyen en el CD ROM que acompaña a este número de la revista.

Así mismo, la evolución entre sus diferentes versiones ha mejorado la seguridad, el manejo de algunos elementos, así como la inclusión de nuevas facilidades como la comunicación por *sockets*.

¿QUÉ ES TCL/TK?

El Tcl (*Tool Command Language*) es un lenguaje de comandos similar a otros ya existentes en UNIX, y a los cuales se les conoce como "*shells*". Entre las *shells* más comunes están el *Bourn Shell* (sh), el *C Shell* (csh), el *Korn Shell* (ksh) y *Perl*. Las *shells* per-

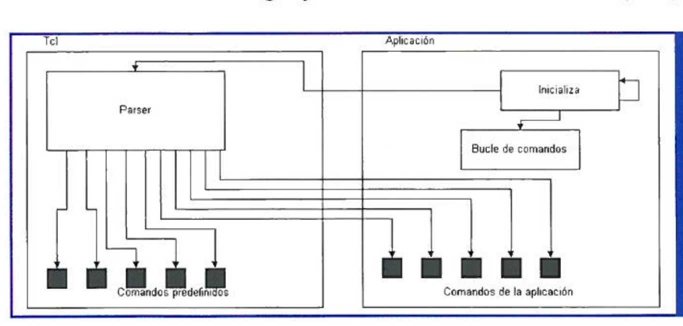


Figura 1. Mecanismo de extensión de comandos de una aplicación a través del Parser de Tcl.

miten ejecutar otros programas, proporcionando suficientes elementos de programación (variables, control de flujo y procedimientos), lo que les da la capacidad de escribir "*scripts*" complejos que pueden añadir a programas existentes en una herramienta construida a medida. Una *shell* proporciona un lenguaje interpretado que permite controlar todos los aspectos de una aplicación interactiva. Esto convierte a las *shells* en un mecanismo formidable para la automatización de rutinas.

Las aplicaciones interactivas tienen la peculiaridad de necesitar la utilización de lenguajes de comandos y normalmente cada aplicación se crea el suyo propio, lo que da como resultado sistemas débiles y peculiares. Aquí precisamente estriba la ventaja del Tcl, ya que incluye una biblioteca de comandos en C con características de procedimientos, variables, listas, expresiones, bucles, etc..., además de ser extensible por las aplicaciones.

A diferencia de otros *Shells*, el Tcl es extensible, ya que se le pueden agregar nuevas primitivas o comandos escribiendo los nuevos procedimientos en C (ver Figura 1). Además, la comunidad Tcl ha contribuido con infinidad de nuevos comandos que se pueden agregar directamente.

Las aplicaciones creadas en C generan nuevos *scripts* Tcl, que son analizados por el *Parser* de Tcl, el cual llama a los procedimientos asociados a cada comando. Por lo tanto, la aplicación extiende los comandos predefinidos por Tcl, con lo que se obtienen:

- Nuevos tipos de objetos en C.
- La implementación de primitivas sobre los objetos como nuevos comandos Tcl.
- La creación de comandos complejos como comandos Tcl.

Hay muchas extensiones para Tcl de libre distribución disponibles en Internet. Muchas de ellas incluyen su biblioteca en C, la cual proporciona nuevas funcionalidades. La extensión de Tcl más notable es el Tk, un "*toolkit*" para X-Windows con el cual a través de comandos Tcl se pueden crear y manipular los *widgets* del interfaz de usuario. Y con esto se obtienen ventajas como la simplificación del nivel de programación del ambiente gráfico X-Windows, con lo que se aumenta la velocidad de desarrollo, con el beneficio de que más cosas son programables y teniendo colaboración entre aplicaciones.

Tcl/Tk un entorno de programación para la creación de interfaces de usuario para X-Windows como para Windows 95/NT

DESARROLLO DE INTERFACES

Las aplicaciones basadas en ventanas son difíciles de programar debido a la complejidad añadida por la parte gráfica de la plataforma en la que se desarrollan, ya que las bibliotecas de funciones presentan una gran complejidad. Sin embargo, la tendencia general en cuanto a herramientas gráficas es reducir los tiempos de desarrollo de los interfaces gráficos, lo cual ha provocado el gran éxito de herramientas *RAD* como Visual Basic y Delphi en la programación para Windows. Así pues, el *toolkit* Tk guarda cierta similitud con estas herramientas, ya que facilita de igual manera la construcción de entornos gráficos.

Para mostrar la simplicidad de la utilización del Tcl/Tk, veamos el típico programa "Hola Mundo":

```
button .hola -text "¡Hola Mundo!" -
command exit
pack .hola
```

Como su puede ver, cada ventana o *widget* tiene su nombre, en este caso *.hola*. Así pues, los *widgets* se crean con el comando "nombre de la clase":

```
button .hola -text "¡Hola Mundo!" -
command exit
```

Además, de haberse creado el comando *.hola*, también se le pueden indicar al gestor las coordenadas donde situar el *widget*. Para el ejemplo anterior sería:

```
place .hola -x 10 -y 25
pack .hola -side left
```

De igual manera, se pueden manipular los *widgets*:

```
.hola flash
.hola configure -foreground red
```

En el primer caso el botón *.hola* parpadeará, y en el segundo pone el botón de color rojo.

De igual manera, los *widget* se interconectan entre sí. Un botón llama a un *script* Tcl, un *scrollbar* y una *listbox* se comunican mediante Tcl, se puede definir la gestión de eventos (*bind*), o hacer selecciones vía Tcl, lo cual se puede ver en la Figura 3.

```
scrollbar .scroll -command ".list
yview"
pack .scroll -side right -fill y
listbox .list -yscroll ".scroll set"
```

Tcl/Tk es fácil de aprender, extremadamente poderoso y contiene muchos elementos sofisticados

```
button .hola -text "¡Hola Mundo!" -command exit
pack .hola
```



Figura 2. Interfaz gráfica del programa "Hola Mundo"

```
scrollbar .scroll -command ".list yview"
pack .scroll -side right -fill y
listbox .list -yscroll ".scroll set"
pack .list -side left
bind .list <Key-q> {puts adios; destroy .}
bind .list <Double-1>{mover[selection get]}
focus .list
proc llenar {} {
        global dir
        .list delete 0 end
        foreach i [exec ls -aF $dir] {
                .list insert end $i
        }
}
proc mover {marca} {
        global dir
        if {[file isdirectory $dir/$marca]} {
                set dir $dir/$marca
                llenar
        }
}
set dir "."
llenar
```

```
llenar.tcl
./
../
c.ps*
cps~1.gz*
drawing1.vsd*
figura0.bmp*
figura1.bmp*
figura2.bmp*
figura3.bmp*
hellow~1.bak*
```

Figura 3. Navegador de directorios realizado en Tcl/Tk.

```
pack .list -side left
bind .list <Key-q> {puts adios; destroy .}
bind .list <Double-1>{mover[selection get]}
focus .list
proc llenar {} {
        global dir
        .list delete 0 end
        foreach i [exec ls -aF $dir] {
                .list insert end $i
        }
}
proc mover {marca} {
        global dir
        if {[file isdirectory $dir/$marca]} {
                set dir $dir/$marca
                llenar
        }
}
set dir "."
Llenar
```

COMPOSICIÓN DE APLICACIONES

En ocasiones es necesario construir sistemas en los que distintas aplicaciones tengan que comunicarse entre sí, ya sea para el intercambio de datos o para que se realice cierta acción. Por ejemplo, el envío de comandos a un editor para resaltar una cierta línea de ejecución, o en un panel de control que invoca y utiliza varias aplicaciones. En particular, el Tcl/Tk, a través de sus primitivas "*send*" y "*socket*" o de algunas extensiones como las proporcionadas por el Tcl-DP, el cual es bastante potente en este sentido. Además, el Tcl/TK cuenta con un mecanismo de seguridad ciertamente aceptable y con el que se pueden construir sistemas complejos a partir de hiperherramientas especializadas y reutilizables u objetos activos a través de comandos Tcl enlazados, como hipertexto o hipermedia.

TCL/TK CON INTERNET

Debido a sus virtudes en seguridad y a su facilidad de programación, el lenguaje Tcl/Tk ha ido haciéndose un lugar entre los lenguajes de programación de Internet, junto a otros lenguajes populares como *Perl* o Java. De hecho, ya existen *plug-ins* para Netscape y Explorer con los que se pueden visualizar aplicaciones similares a los *applets* de Java, y que se han bautizado como *Tclets*. En la Para la inclusión de un "*tclet*" en una página html sólo es necesario declarar el objeto incrustado tal como sigue:

```
<embed src=tktetris.tcl width=300 height=500>
```

sólo habrá que considerar que algunos comandos no son permitidos en los "*tclets*".

INCONVENIENTES DEL TCL/TK

Debido a que es un lenguaje interpretado, tiene un límite en las prestaciones de velocidad que puede ofrecer. Sin embargo, se pueden obtener resultados sorprendentes. Y para aquellos que ya tengan sistemas construidos con *Xt* o con la librería *Motif* no les será de mucha ayuda, ya que es incompatible con estos sistemas. Otro inconveniente que pude retardar el diseño de una aplicación es que no analiza la semántica estática de su código, por lo que los errores solamente serán detectables en el momento de ejecución.

CONCLUSIONES

Como se ha visto, Tcl/Tk ofrece una programación de alto nivel que requiere poco tiempo de aprendizaje, y donde la velocidad de desarrollo es muy alta. Con él se pueden programar aplicaciones multiplataforma de gran complejidad, las cuales puedan comunicarse entre sí, además de proporcionar un buen nivel de seguridad que lo ha llevado a hacerse un hueco en la programación de aplicaciones en Internet.

BIBLIOGRAFÍA

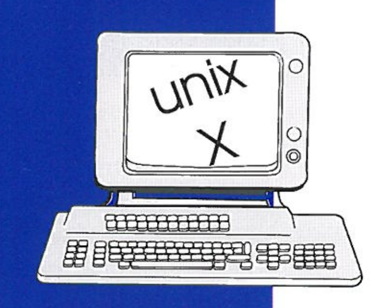
John K. Osterhout, "Tcl and the Tk Toolkit", Addison Wesley, 1994

Brent B. Welch, "Practical Programming in Tcl and Tk", Prentice Hall, 1995

News: comp.lang.tcl

INTERNET: CONMUTACIÓN DE PAQUETES

Fernando J. Echevarrieta



En los últimos artículos se ha procedido a mostrar el funcionamiento de redes TCP/IP mediante el estudio de estos protocolos. Ya al hablar de IP se comentaba que el encaminamiento era una de las funciones principales de un nivel de red. En este número se estudiarán las distintas formas de hacer que un paquete alcance su destino.

A lo largo de anteriores entregas se ha explicado el funcionamiento de redes TCP/IP como Internet, exponiendo la descomposición en niveles de esta arquitectura y procediendo al posterior análisis de los protocolos IP y TCP. En este último caso, se aprovechó la ocasión para presentar otros muchos conceptos relacionados con detección y tratamiento de errores, negociaciones, etc... En los artículos dedicados a IP se comentaba el formato de los datagramas IP, se planteaban las causas de la necesidad de realizar fragmentación de PDUs y cómo solucionaba IP este problema. Ya entonces se estudió cómo cada datagrama indicaba un origen y un destino. En todo momento se tenía presente que la función del nivel inter-red o nivel 3 era generar una red virtual que permitiera comunicarse entre sí a máquinas que no se encontraban directamente conectadas. Por ello, la función de encaminamiento era función clave en este nivel. Se comentó que IP seguía un encaminamiento basado en tablas, e incluso se mencionaron algunos protocolos para la transmisión de información de encaminamiento. Sin embargo, hasta ahora no se ha justificado cómo funciona realmente el encaminamiento en una red de conmutación de paquetes. Este será el tema principal del presente artículo.

Por otra parte, al igual que el mes pasado se explicaban algunas técnicas que permitían controlar el flujo en un enlace, para que una red funcione correctamente será necesario realizar un control global de la congestión. Este será el segundo punto a tratar este mes.

Como se acostumbra en esta sección, se abordarán los temas desde la perspectiva más amplia posible, para lo que, aunque se esté hablando de redes de conmutación de paquetes, previamente se enmarcarán éstas en el contexto de las redes de telecomunicación comparándolas con otras redes, especialmente con las de conmutación de circuitos, y explicando cómo se pueden emular estas últimas mediante la generación de circuitos virtuales.

REDES DE TELECOMUNICACIÓN

Existen innumerables tipos de redes de telecomunicación, por lo que a la hora de clasificarlas es necesario fijar ciertos criterios clave. Atendiendo a la forma de realizar la comunicación entre sistemas finales, las redes de telecomunicación se dividen en dos grandes grupos: redes de conmutación y redes de difusión.

Una red de conmutación es aquella que se encarga de proporcionar el camino adecuado para la información entre origen y destino mediante conmutaciones, por lo que la clasificación se puede llevar más allá distinguiendo entre redes de conmutación de circuitos y redes de conmutación de paquetes. En el primer caso, el problema fundamental consistirá en el establecimiento de un circuito que comunique origen y destino, mientras que en el segundo el problema será sutilmente diferente y consistirá en hacer posible que cada unidad de información sea encaminada del modo correcto hacia su destino, eligiendo entre las posibles alternativas que se le presenten en cada salto.

Una red de difusión, en cambio, es aquella red en la que existe un medio

compartido por emisores y receptores. El emisor coloca la información en el medio compartido y son los receptores los encargados de seleccionar cuál es la información que deben tomar y cuál no. Los casos más claros son los de redes de transmisión por radio, por ejemplo, redes de radiopaquete o redes VSAT por satélite. Sin embargo, una simple red local ethernet corresponde también a este grupo.

En el caso de las redes de difusión no existe el problema de encaminar la información entre origen y destino. Al encontrarse ambos en un medio común, se "escribe" y se "lee" en este medio. En este tipo de redes son otros los problemas a resolver. Del lado de la fuente, el tratamiento de colisiones, que se producen cuando dos fuentes emiten en el mismo medio a la vez interfiriéndose la una a la otra y destruyéndose ambos mensajes. Será necesario detectar y tratar este fenómeno. Del lado del receptor surge el problema de la selección de la información.

REDES DE CONMUTACIÓN

En el estudio de la transmisión de información a través de una red conmutada se pueden distinguir tres etapas diferenciadas: establecimiento de conexión, transferencia de información y liberación de conexión. La única de estas etapas que siempre es necesaria es, por supuesto, la transferencia de información ya que, como se vio en otros artículos de la serie, no todas las comunicaciones llevan asociada una conexión.

CONMUTACIÓN DE CIRCUITOS

En una red de conmutación de circuitos, la fase de establecimiento comienza mediante la identificación por parte del origen del primer nodo intermedio, al que se envía la petición de conexión. Éste actúa del mismo modo con el siguiente y así sucesivamente hasta llegar al destino. El conjunto de nodos por los que va pasando la petición va configurando el circuito físico entre origen y destino y por él retorna la aceptación del destino. Es el caso de la red telefónica convencional, que por ello recibe el nombre de RTC o Red Telefónica Conmutada. En una red de estas características, cuando no es posible encontrar un camino entre origen y destino por ocupación de los nodos intermedios la conexión debe ser rechazada, pero si el circuito se establece es usado exclusivamente para la comunicación que generó la conexión hasta que ésta termine.

GRAFO DE ENCAMINAMIENTO DE UNA RED

Figura 1. Los algoritmos de encaminamiento estudian un grafo que representa los enlaces presentes en la red con su métrica asociada en cada instante de tiempo.

Así pues, la transferencia de información se realiza de forma bidireccional a través de una línea dedicada mientras dure la comunicación, por lo que no existen problemas de encaminamiento ni congestión.

La liberación de la conexión se realiza cuando el nodo origen envía la petición correspondiente.

CONMUTACIÓN DE PAQUETES

Una red de conmutación de paquetes puede contemplar dos modos de operación bien diferenciados: la comunicación mediante datagramas o la generación de circuitos virtuales.

En una red de datagramas no existen las fases de establecimiento y liberación de conexión ya que, de hecho, no existe conexión.

La transferencia de información se lleva a cabo mediante el envío de datagramas, paquetes en cada uno de los cuales se indica origen y destino. Cada uno de los nodos, desde el origen, pasando por los intermedios, hasta el final, debe únicamente conocer cuál sera el siguiente nodo al que "pasar la bola", proceso que, en sí mismo, constituye la función de encaminamiento. Cada nodo intermedio reenvía cada datagrama que recibe sin almacenarlo, y cada datagrama es encaminado de forma independiente. Por ello, las rutas de los paquetes pueden ser diferentes, por lo que pueden llegar desordenados. Este es el caso, como ya se vio en anteriores artículos, de redes como Internet, basada en el protocolo IP. Otro ejemplo clásico es el del correo postal ordinario. En este tipo de redes, al no emplearse conexiones que pueden ser aceptadas o rechazas será necesario realizar un control de congestión que asegure que la red facilita el servicio que debe.

Pero existe una forma alternativa de funcionamiento de las redes de conmutación de paquetes, que consiste en emular las redes de conmutación de circuitos mediante el establecimiento de circuitos virtuales. En este caso, se establece una conexión entre origen y destino de forma análoga a como se iba construyendo el circuito físico en el caso anterior. El nodo origen identifica al primer nodo intermedio y envía un paquete de petición. Este proceso se va repitiendo hasta llegar al destino. Si se produce aceptación en destino, habrá quedado establecido un circuito virtual entre todos los nodos intermedios por el que circularán todos los paquetes correspondientes a la comunicación.

La transferencia de información se realiza enviando cada paquete al siguiente nodo del circuito virtual. Se trata de un circuito virtual, ya que todos los paquetes involucrados en la misma comunicación pasarán exactamente por el mismo camino, por lo que se comportarán del mismo modo que si recorrieran un circuito físico y llegaran ordenados a destino. Para que esto suceda así, cada paquete lleva asociado un identificador de circuito virtual en

TABLAS DE ENCAMINAMIENTO

Tabla de encaminamiento genérica (abstracción)

A		
Nodo	Distancia	Enlace Salida
A	0	-
B	3	B
C	1	C

Tabla de encaminamiento en Linux (tal y como la muestra el comando route)

Destination	Gateway	Genmask	Flags	MSS	Window	Use	Iface
138.4.22.0	*	255.255.255.192	U	1500	0	5420	eth0
loopback	*	255.0.0.0	U	3584	0	14	lo
default	isabel.dit.upm.	*	UG	1500	0	7661	eth0

Figura 2. Todos los nodos de una red disponen de una tabla de encaminamiento como las que se representan en la figura. En el primer caso se considera una tabla genérica en el que a cada nodo se asocia una salida y se indica su métrica. En el segundo, se observa el caso particular de Linux, en el que no aparece la métrica por no encontrarse aún implementada en la versión del ejemplo. En él también se observan rutas por defecto.

función del cual cada nodo decide cuál será el siguiente salto. La diferencia con el empleo de circuitos reales es que, en este caso, los enlaces entre nodos no son dedicados, ya que pueden ser compartidos por varios circuitos virtuales. Esto supone la ventaja de un mayor aprovechamiento de la infraestructura de comunicaciones ya que, aunque no se utilice un circuito virtual establecido, los enlaces podrán ser utilizados por otras comunicaciones. La desventaja es que, al no ser dedicado, es mucho más difícil garantizar la calidad de servicio de cada comunicación. Por ello, en este caso, será necesario emplear técnicas de control de congestión.

A pesar de que una red funcione internamente basada en datagramas, puede ofrecer una interfaz que proporcione un servicio que simule circuitos virtuales. Para ello, la propia red debe encargarse de reordenar los paquetes en destino antes de entregarlos.

ENCAMINAMIENTO EN REDES DE CONMUTACIÓN DE PAQUETES

En los artículos dedicados a IP se definió la función de encaminamiento y se presentó como una de las principales, si no la principal, función de un nivel de red. En este caso, se introducirán los distintos tipos de encaminamiento que se pueden encontrar en una red de conmutación de paquetes. La primera división a realizar en cuanto a los tipos de encaminamiento es la clasificación en encaminamiento estático y encaminamiento dinámico. En el primero de los casos, cada nodo dispone de una tabla fija de rutas independiente del estado de la red, mientras que en el segundo cada nodo examina el estado de la red y decide la ruta en cada caso.

Así pues, en encaminamiento dinámico o adaptativo es necesario realizar una toma de decisiones. Por ello, además de una tabla de encaminamiento en cada nodo, es necesario disponer de un algoritmo de encaminamiento. Este algoritmo se utilizará para determinar el mayor o menor grado de bondad de cada enlace que pueda servir como camino de salida. Para ello es necesario definir una métrica asociada a cada enlace, que recibe el nombre de coste o distancia (figura 1). La función de coste de cada enlace dependerá de diversos factores, como el propio coste económico de utilización, los factores de calidad de servicio del enlace, latencia, ancho de banda, grado de congestión, etc... Todos estos factores pueden variar con el tiempo, por lo que el encaminamiento puede también varia. De ahí que se hable de encaminamiento dinámico.

Una red que emplea encaminamiento dinámico funciona de modo más eficiente al adaptar sus rutas dinámicamente en función del estado de la red. Sin embargo, como en todo, cada ventaja tiene su precio. Así, este tipo de encaminamiento supone un mayor consumo de recursos no sólo en tiempo de CPU y memoria en los *routers*, sino también en ancho de banda de la red al ser necesario, en la mayoría de los casos, un intercambio de información entre nodos que refleje los cambios en el estado de la misma.

Existen tres grupos principales de clasificación de los métodos de encaminamiento adaptativo: encaminamiento aislado, encaminamiento centralizado y encaminamiento distribuido.

ENCAMINAMIENTO AISLADO

Con los métodos de encaminamiento aislado, cada *router* funciona de modo independiente sin que exista comunicación con los demás. Sin embargo, no tiene una tabla de encaminamiento estática, sino que va adaptándose a las condiciones de la red. Así, dentro de este grupo se emplean fundamentalmente tres algoritmos: el de la patata caliente, el de inundación y el de aprendizaje hacia atrás.

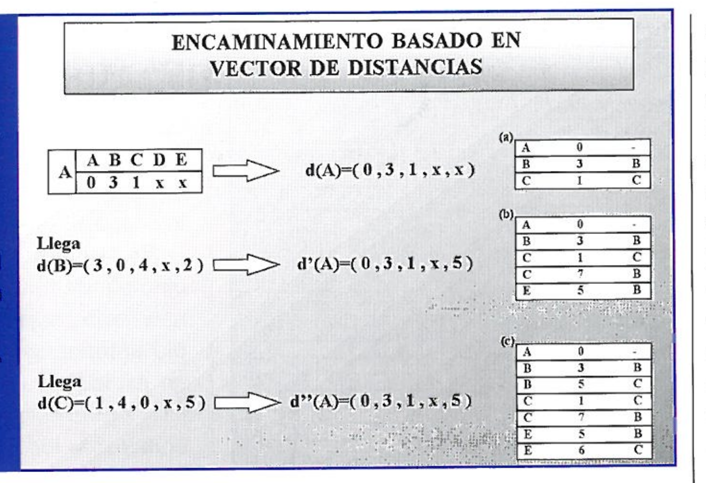


Figura 3.
Cuando se emplea encaminamiento basado en vector de distancias el tamaño de los mensajes depende del tamaño de la red, pero el número de mensajes que envía cada nodo es constante.

El algoritmo de la patata caliente, recibe este curioso nombre debido a su comportamiento. Si a uno le pasan una patata tan caliente que queme, tratará de pasarla a otro por cualquier medio, pero lo antes posible. Así, este algoritmo no emplea tablas de encaminamiento sino que, cuando recibe un paquete, trata de librarse de él lo antes posible, por lo que lo coloca en aquella cola de salida que se encuentre más vacía. De esta manera, los paquetes siempre salen por el enlace que dispone de un menor tráfico, el que menos congestionado esté. Así, el tráfico se balancea en la red. El problema es que con este algoritmo no existe ninguna garantía de que, al final, el paquete llegue a su destino. De hecho, se pueden producir bucles, no hay forma de predecir el retardo y se produce desorden.

El algoritmo de inundación sigue una filosofía completamente distinta, aunque tampoco emplea tablas de encaminamiento. Cuando llega un paquete, simplemente se le coloca en todos los enlaces de salida. De esta forma, esta claro que llegará a destino, ya que irá por todos los caminos posibles. Además al recorrer todos los caminos, también está claro que una copia irá por el mejor de los caminos posibles. Pero tampoco parece la mejor de las ideas, ya que la avalancha de duplicados produce, como su nombre indica, una inundación de la red. Este algoritmo no es en realidad un algoritmo adaptativo, ya que siempre se comporta igual. Sin embargo, sirve de base para el de aprendizaje hacia atrás, que se describe a continuación.

Por último, el algoritmo de aprendizaje hacia atrás es el más razonable dentro de este grupo. Cuando comienza a funcionar tampoco dispone de una tabla de encaminamiento, o mejor dicho, dispone de una, pero vacía. Al recibir los paquetes, funciona al principio como el algoritmo de inundación, copiándolos a todos los enlaces, con lo que se garantiza que, salvo apertura en la red, el paquete llegará a destino. Pero ahora, el algoritmo, va apuntando en la tabla el lugar por el que viene cada paquete, lo que ya le va dando una idea de la localización de cada máquina. Las entradas que realiza en la tabla suelen ir asociadas a un *time-out*, por lo que las rutas que va aprendiendo pueden ir variando con el tiempo, con lo que se produce su adaptación a la evolución del estado de la red.

ENCAMINAMIENTO CENTRALIZADO

El encaminamiento centralizado se fundamenta en la existencia de una serie de tablas dinámicas en cada nodo que responden a un control central. Así, cada nodo intermedio envía información periódicamente a este control central de red que dispone de la visión que de la misma tienen todos los nodos y, en función de esta información, genera las tablas que, posteriormente, distribuye a cada nodo. La principal ventaja de este mecanismo en comparación con los métodos de encaminamiento aislado es que, de esta forma, las tablas se construyen partiendo de una visión global del estado de la red, y no de los enlaces que rodean a un solo nodo, como ocurría con el aprendizaje hacia atrás. Por otra parte, el software en los *routers* se hace muy simple, ya que toda la complejidad reside en el control central. El problema principal surge también de esta característica. El punto débil de todo el encaminamiento de la red es este control central. Si cae, la adaptabilidad del encaminamiento desaparece y si produce errores, fallarán todas las rutas de la red. Por otra parte, al enviar y recibir información de control de todos los nodos, todo el tráfico de control pasará por sus enlaces, creando una zona a su alrededor de mayor congestión. Esto, como se verá más adelante, acrecienta el problema de que no todas las tablas se recibirán a la vez en los *routers*, por lo que es posible que, en el mismo instante, algunos dispongan de una tabla actualizada mientras que otros deban aún actualizarla. Esta clase de inconsistencias puede dar lugar a la aparición de bucles temporales en la red.

Si bien se cuenta con estos problemas, se puede apreciar una mejora respecto a los métodos de encaminamiento aislado, aunque las soluciones más empleadas surgen de un tratamiento distribuido de la información de encaminamiento en la red.

ENCAMINAMIENTO DISTRIBUIDO

Los algoritmos de encaminamiento distribuido se clasifican en dos grandes grupos: algoritmos basados en vector de distancias y algoritmos basados en estado de enlaces.

En redes que emplean algoritmos de construcción de vector de distancias los nodos intermedios no disponen de una visión global de la red, sino únicamente de un entorno cercano. Con este mecanismo, cada nodo intercambia la información de estado de sus enlaces con los nodos adyacentes. Al comenzar el algoritmo, cada *router* dispone de una tabla como la que se puede observar en la figura 2. Para obtener más información sobre la tabla concreta de Linux y UNIX consúltese [Echeva-1]. En el ejemplo, en cada entrada se asocia a un destino una distancia, métrica o coste, y un enlace de salida. Estas entradas se pueden considerar como un vector que indica la distancia asociada a cada enlace, de donde toman los algoritmos el nombre. Cada *router* envía esta tabla de métricas a los nodos adyacentes (y sólo a los adyacentes) y a su vez compara la

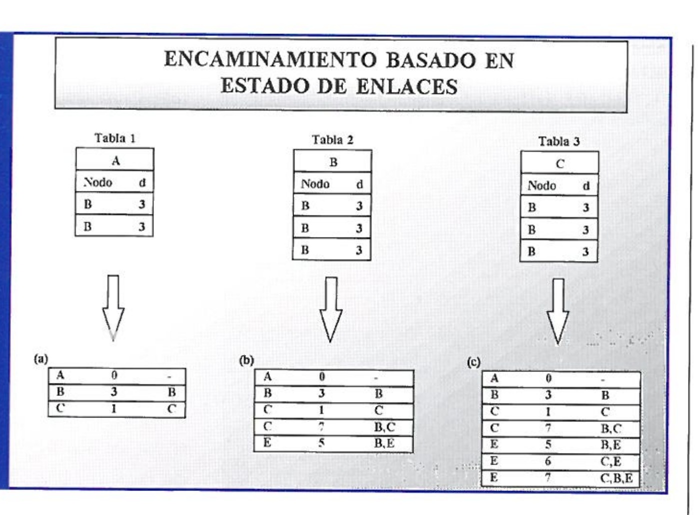


Figura 4. Cuando se emplea encaminamiento basado en estado de enlaces, el tamaño de los mensajes que envía cada nodo es constante, pero su número depende del tamaño de la red.

que tiene con las que recibe de sus vecinos, modificándola para contemplar la nueva información. El número de mensajes de encaminamiento que debe enviar cada nodo en cada actualización es constante, ya que sólo intercambia información con los nodos contiguos. Sin embargo, el tamaño de los mensajes dependerá del tamaño de la red.

En la figura 3 se puede observar un ejemplo de encaminamiento basado en vector de distancias. Como ejemplo se ha partido de la red compuesta por cinco nodos que se indicaba en la figura 1 y se estudia lo que sucede en el nodo A. En la figura se han representado los vectores de distancias de los nodos A, B y C, que se han denominado d(A), d(B) y d(C) respectivamente. Estos vectores de distancias, que sólo se transmiten a los nodos vecinos, son de tamaño constante y su dimensión es igual al número de nodos de la red. Al comienzo, la tabla de encaminamiento del nodo A sería la representada en (a). Si en un instante recibe el vector de distancias de B, d(B), pasará a modificar su propio vector de distancias quedando como d(A) y podrá ampliar su tabla de encaminamiento quedando ésta como en (b). En este primer movimiento ya vemos cómo se detecta una ruta alternativa para llegar a C, ya sea a través de B o directamente. La distancia coste o métrica asociada a cada uno de los enlaces es de 1 y 7 respectivamente, con lo que, en ese momento, lo lógico es que los paquetes hacia C vayan directamente. Sin embargo, si con el tiempo se rompe ese enlace (con lo que su distancia pasaría a ser infinita) o se congestiona por cualquier razón, su distancia puede ser superior a la que se asocia a la ruta alternativa, con lo que el encaminamiento variaría.

Si tras ello se recibiera el vector de distancias de C, vemos cómo ya no variaría el vector de distancias de A, aunque se dispondrá de la información de las distancias asociadas a cada uno de los dos enlaces de que dispone el nodo.

Los vectores de distancias irán actualizándose y variando dinámicamente. Así, cuando B o C reciban información de E y transmitan sus vectores de distancias modificados a A, la distancia de A a D dejará de ser considerada infinita. En cualquier caso, A sólo podrá decidir cuál es el siguiente nodo más conveniente para llegar a un destino, B o C, pero jamás dispondrá de información sobre todos los enlaces de la red y, por tanto, nunca podrá realizar un encaminamiento en origen. Es decir, el encaminamiento tendrá lugar salto a salto y sólo se podrá decidir el enlace de salida más apropiado.

Los algoritmos basados en el estado de enlaces, en lugar de intercambiar información con los nodos contiguos, envían la información del estado de sus enlaces a todos los nodos de la red. De esta forma, la convergencia de las tablas de encaminamiento se produce de forma más rápida que con los algoritmos de vector de distancias. El tamaño de la información correspondiente a los enlaces de cada nodo es constante, por lo que el tamñao de los mensajes enviados no depende del tamaño de la red. Sin embargo, es necesario enviar más mensajes de encaminamiento que con los algoritmos de vector de distancias, ya que esta información no se comunica únicamente a los nodos contiguos. Por otra parte, con este método cada nodo recibe información del estado de toda la red, no sólo de su entorno cercano como ocurría en el caso anterior. Por ello, este mecanismo permite realizar encaminamiento fijado en origen que, como se vio en [Echeva-2], se le puede indicar a IP mediante opciones.

En la figura 4 se puede observar un ejemplo de funcionamiento de encaminamiento sobre la misma red del ejemplo anterior, pero en este caso basado en estado de enlaces. En la figura se han representado los estados de los enlaces de los nodos A, B, y C, que se indican mediante tablas numeradas como 1, 2 y 3 respectivamente. La información contenida en estas tablas es lo que se transmite a todos los nodos de la red. Así, la tabla de encaminamiento del nodo A al comienzo será la representada en (a). Si en un instante recibe la información del estado de enlaces de B, tabla 2, pasará a ampliar su tabla de encaminamiento, quedando ésta como en (b). En este primer movimiento también se detectan dos posibles rutas para E. Al recibir el nodo A la información del estado de enlaces de C, tabla 3, podría ser aumentando su información de encaminamiento para construir una tabla como (c). Nótese que esta tabla puede contener información sobre rutas completas y no únicamente sobre enlaces de salida, ya que en A se podría ir reconstruyendo completamente el grafo de métricas de la red, de tal modo que, una vez recibida la información de todos los nodos se dispondría del grafo completo, lo que permitiría realizar encaminamiento en el origen. Baste imaginar, observando el ejemplo, que cuando el nodo A recibiera el estado de enlaces de E obtendría todas las rutas de la red, lo que jamás habría ocurrido aplicando vector de distancias.

CONTROL DE CONGESTIÓN

En redes de conmutación de paquetes es necesario llevar a cabo un control de congestión. Al contrario que el control de flujo, que es un problema local en el que se trata de evitar la saturación del enlace que interviene en la comunicación entre dos máquinas, en el caso del control de congestión se trata con un problema global que afecta al comportamiento de todos los nodos de la red.

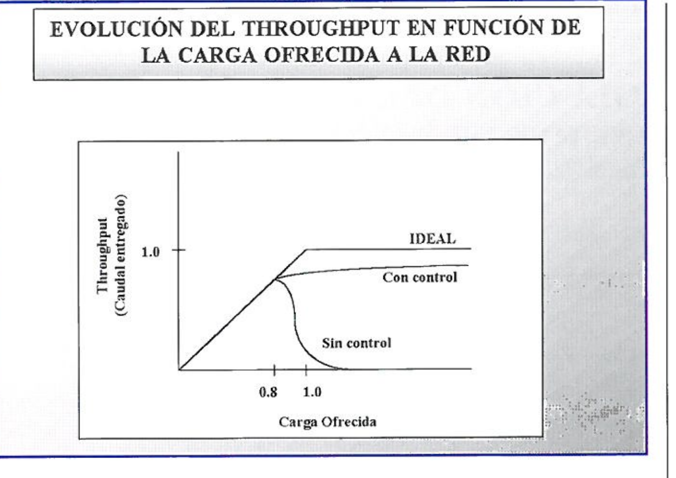


Figura 5.
Al contrario que una tubería, si se trata de transmitir más caudal por una red real que el que ésta permite, en lugar de trabajar ofreciendo su máximo caudal teórico, la red se colapsa y su throughput cae drásticamente.

Mediante el control de congestión de una red se garantiza que ésta pueda transportar todo el tráfico ofrecido y de esta forma responder a unos requisitos de QOS (*Quality Of Service*) o calidad de servicio.

En las figuras 5 y 6 se puede observar cómo evoluciona el *throughput* (caudal entregado) de una red y el retardo medio de cada paquete, respectivamente, en los casos ideal, con control de congestión y sin control de congestión. En este último caso se puede apreciar cómo a partir de una carga de 0.8 el sistema comienza a colapsarse, se disparan las colas por nodo, se rechazan cada vez más paquetes, los nodos se ven obligados a producir reenvíos que, a su vez, colapsan más la red, y el retardo medio por paquete se dispara a la vez que se produce una caída del *throughput*. En la situación de congestión se produce el colapso y el sistema no funciona aunque todos sus recursos se encuentran operando al máximo. Por ello, es necesario controlar la congestión para que no se supere un umbral de carga.

Al igual que para el tratamiento de errores, como se vio en [Echeva-3], exiten dos formas de abordar el problema del control de la congestión de una red. Por una parte, anticipándose a la congestión mediente el empleo de mecanismo preventivo y, por otra, resolviéndola del modo más eficiente posible, una vez se ha producido, mediante mecanismos correctores.

El algoritmo de la patata caliente trata de librarse cuando antes de un paquete, como si quemara

MECANISMO DE RESERVA DE BUFFERS

El mecanismo de reserva de *buffers* se emplea en redes que establecen circuitos virtuales como un medio de garantizar cierta calidad de servicio si se acepta una comunicación o de rechazarla antes de contribuir a una mayor congestión de la red.

Durante la fase de establecimiento del circuito, cada sistema intermedio reserva *buffers* para cada circuito en cada sentido. Cuando se solicita el establecimiento de un nuevo circuito virtual se comprueba si quedan *buffers* libres en todos los sistemas intermedios del recorrido y, si no es así, se rechaza la comunicación notificando la condición de ocupación de la red. De esta forma, el procedimiento de aceptación o rechazo de una comunicación es idéntico al que se realiza en una red de conmutación de circuitos.

Los asentimientos a cada paquete se realizan cuando éste sale del nodo que lo recibió con destino al siguiente nodo intermedio, en lugar de realizarse a la recepción del mismo. De esta manera, si se recibe un *ACK* queda garantizado que existe un *buffer* libre para recibir el siguiente paquete. Esto determina el flujo en los enlaces con protocolos de parada y espera [Echeva-3].

El principal problema que presenta este mecanismo es que, al igual que en una red de conmutación de circuitos, en que los circuitos quedan dedicados, en la red pueden existir circuitos reservados que no se usan, lo que supone una infrautilización de la misma. Por tanto, este mecanismo únicamente es aconsejable en aquellos casos en que se necesite cierto ancho de banda o tiempo de respuesta en las comunicaciones. Aún así, este problema no es tan grave como con circuitos reales porque los enlaces, que en este caso no son dedicados, podrán seguir siendo utilizados por otros circuitos virtuales.

MECANISMO DE DESCARTE DE PAQUETES

Cuando se emplea el mecanismo de descarte de paquetes no se reserva ningún *buffer* para la comunicación, de tal forma que si llega un paquete y no existe sitio para él en el nodo intermedio, simplemente se elimina. Como se ha visto en artículos anteriores, esto no es mayor problema en redes no fiables de datagramas. Sin embargo, en aquellas redes que establecen circuitos virtuales exige el mantenimiento de copias de cada paquete con el objeto de poder retransmitirlas. Para ello, el nodo transmisor del paquete descartado espera el vencimiento de un *time-out* y procede a retransmitirlo.

El inconveniente de este sistema es que si el *time-out* que activa una retransmisión se ha ajustado de tal forma que supone un tiempo inferior al que la posible congestión de la red impone para el retorno del asentimiento, el origen volverá a retransmitir un paquete que no sería necesario retransmitir, aumentando, aún más, la congestión de la red. Por ello, si se emplea esta técnica es fundamental garantizar que un asentimiento jamás sea descartado.

CHOKE PACKETS

Este mecanismo es una técnica combinada preventiva-correctiva. Cada uno de los nodos intermedios mide la utilización de cada canal de salida y,si este valor excede un valor umbral predeterminado, se envía un paquete denominado *choke packet* a la fuente,indicándole la posibilidad de con-

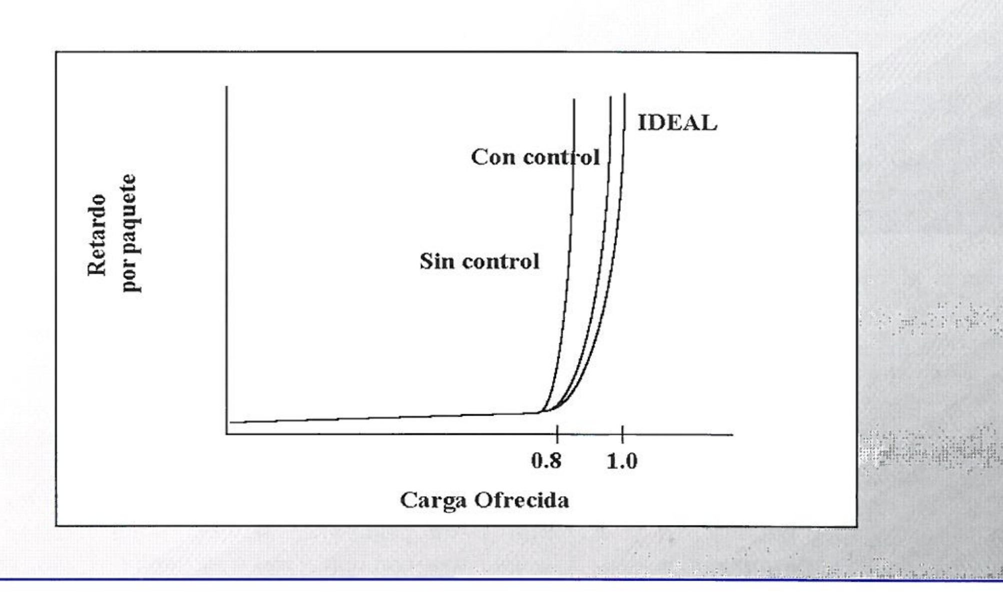


Figura 6: Con cargas cercanas al límite se disparan las colas por nodo, se rechazan cada vez más paquetes, lo que implica nuevos reenvíos que saturan aún más la red y ésta entra en colapso.

gestión. Cuando una fuente recibe uno de estos paquetes, reduce su flujo y comienza a descartar los sucesivos choke packets que recibe. Pasado cierto tiempo de precaución, la fuente comenzará a aumentar su flujo de nuevo de manera progresiva y atenderá nuevamente los posibles *choke packets* que reciba.

Así pues, este mecanismo se puede considerar una extensión del control de flujo a través de red.

Una red que emplea este método de congestión cuenta con numerosos parámetros a ajustar. El primero de ellos consiste en la determinación del valor umbral de flujo de salida a partir del cual debe saltar la alarma de congestión. Otros son los necesarios para determinar el intervalo de espera de una fuenta ya alertada antes de comenzar a atender nuevos *choke packets* y la forma de recuperación del flujo de emisión de esta fuente.

En ocasiones se combina la comunicación de la información de encaminamiento entre nodos con la propia información de congestión.

CONCLUSIONES

Con el presente artículo se ha pretendido proporcionar al lector una visión amplia de los problemas que afectan al funcionamiento de una red de conmutación de paquetes aún cuando ya se hayan resuelto los problemas relacionados con el formato de la información a transmitir. No era intención realizar un recorrido exhaustivo por los distintos algoritmos de encaminamiento, que se basan en el estudio de grafos. Algunos de estos algoritmos, como el de Dijkstra o de Floyd, fueron presentados desde el punto de vista de teoría de grafos en los primeros artículos de la revista [Rubio-1], [Rubio-2]. El encaminamiento en redes de conmutación de paquetes es uno de los ejemplos claros en que se emplea parte de esta teoría. También se ha pretendido destacar como aún resueltos todos los problemas para que dos máquinas "se entiendan" y para que la información llegue de una a otra, el sistema aún no funcionaría si no se controlaran los problemas de congestión.

Con este artículo, se termina el recorrido por los problemas principales que se presentan en los protocolos inferiores, los que constituyen una red de transporte. En próximas entregas se volverá a descender al nivel de implementación exponiendo la programación de accesos a servidores de nombres, algunas técnicas de programación de servidores, empleo de señales, mensajes fuera de banda y algunos trucos como la transmisión de mensajes delimitados a través de TCP. También se continuará con la exposición de los niveles superiores de la torre OSI.

REFERENCIAS

Las referencias citadas corresponden a artículos publicados en Solo Programadores por David Rubio según:

[Rubio-1], "Teoría de Grafos", núm 4

[Rubio-2], "Aplicando Grafos", núm 5
y el autor de este artículo según:

[Echeva-1], "Conceptos y herramientas UNIX para redes TCP/IP", núm. 23

[Echeva-2], "Chequeo a Internet: El protocolo IP", núm 26

[Echeva-3], "Chequeo a Internet: TCP", núm. 28

INTRABUILDER PROFESIONAL 1.0

Fernando de la Villa

Cada vez más gente confía en Internet como una herramienta de trabajo. Los profesionales buscan, sobre todo, información actualizada y de calidad. Por desgracia, en demasiadas ocasiones esto no es así. Esto se debe principalmente a la dificultad, con las tecnologías existentes hasta hace muy poco tiempo, de mantener actualizada una gran cantidad de información en la red. Para solventar ese problema aparece IntraBuilder, un producto especializado en la creación de páginas web que proporcionan acceso a bases de datos. IntraBuilder es un completo sistema de desarrollo basado en *JavaScript*, un estándar del mercado para la creación de páginas web activas. El producto proporciona todo lo necesario, desde la creación de tablas de las bases de datos hasta el diseño de páginas web, pasando por la edición del código en lenguaje *JavaScript*.

Pero, tal y como sugiere su propio nombre, no está limitado a Internet. Las empresas que utilicen o estén pensando en utilizar en sus instalaciones una intranet (pequeña red que utiliza las tecnologías de Internet) también pueden verse beneficiadas. IntraBuilder puede ayudar a la creación de páginas para la introducción en las bases de datos de la empresa de informes de ventas, datos de clientes, pedidos y un largo etcétera. Cualquier empleado con acceso a la red de la empresa sólo necesita un navegador que soporte *JavaScript* para tener acceso a la información. Además, existen opciones de seguridad para restringir el acceso sólo a las personas autorizadas.

CARACTERÍSTICAS

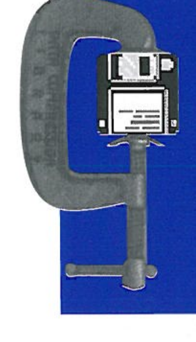
IntraBuilder incorpora multitud de características orientadas a la creación rápida de aplicaciones para Internet. No son necesarios conocimientos de programación ni de los protocolos utilizados en la red para crear páginas web totalmente funcionales. Sin embargo, sólo se puede sacar el máximo provecho del producto mediante el uso de *JavaScript* y las características más avanzadas del lenguaje HTML.

Los servidores web utilizan interfaces específicos para comunicarse con otros programas. Intrabuilder proporciona soporte para los interfaces más comunes: NSAPI de Netscape, ISAPI de Microsoft y CGI (*Common Gateway Interface*). También da soporte directo a los servidores web Netscape FastTrack, Borland Web Server, WebSite (de O'Reilly & Associates) y Microsoft Internet Information Server.

Además, al estar orientado a las bases de datos, incluye soporte para los formatos de Borland (dBase, InterBase y Paradox), de Microsoft (Access y SQL Server) y otros formatos como DB2, Informix, Sybase y Oracle. En general, puede trabajar con cualquier base de datos soportada por ODBC.

En el diseño de páginas web no pueden faltar los gráficos. IntraBuilder admite los formatos GIF, JPEG, BMP, DIB, TIF, PCX, Windows Metafile (WMF), PostScript encapsulado (EPS) y XBitmap (XBM).

Intrabuilder dispone de un entorno integrado de desarrollo (IDE) que cen-



Intrabuilder es una interesante herramienta de creación de aplicaciones para intranets y la World Wide Web. En el presente artículo se analizan las características de su edición profesional.

traliza todas las operaciones relacionadas con la construcción de aplicaciones. Desde el mismo se pueden crear tablas de las bases de datos, formularios, informes, además de editar *scripts* y ejecutar las aplicaciones antes de instalarlas en el servidor web.

Para facilitar el trabajo se incluyen varias utilidades que reciben el nombre de expertos y diseñadores. Los expertos guían paso a paso al usuario al realizar cada tarea, proponiendo valores apropiados por defecto. También ofrecen ayuda en línea sobre las opciones disponibles. Esta es la alternativa más sencilla para crear el esqueleto de una aplicación. La otra posibilidad es utilizar los diseñadores, que permiten una mayor libertad al usuario. En muchos casos se utilizarán los dos. Primero se crea un esbozo de la aplicación con los expertos y a continuación se emplean los diseñadores para ampliar y personalizar el resultado. Estos últimos incluyen características más avanzadas que los expertos, como paletas y posibilidad de arrastrar y soltar objetos.

Con IntraBuilder se pueden diseñar visualmente todos los elementos que forman parte de la aplicación orientada a la web. Se trabaja de una forma similar a Delphi y Visual Basic, con inspectores de objetos y paletas de componentes. Los objetos pueden ser colocados y dimensionados en la pantalla, viendo el resultado tal y como va a aparecer en la realidad. No todas las características comentadas están presentes en todas las ediciones de IntraBuilder. En el mercado se encuentran disponibles tres ediciones diferentes que se ajustan a las necesidades de cada usuario:

- **IntraBuilder:** Es la edición básica. Incluye el IDE y solamente una instancia del servidor IntraBuilder para bases de datos Access, Paradox y dBase. También trae el servidor web de Borland. Está recomendada especialmente para los usuarios que desean dar acceso a pequeñas cantidades de información.

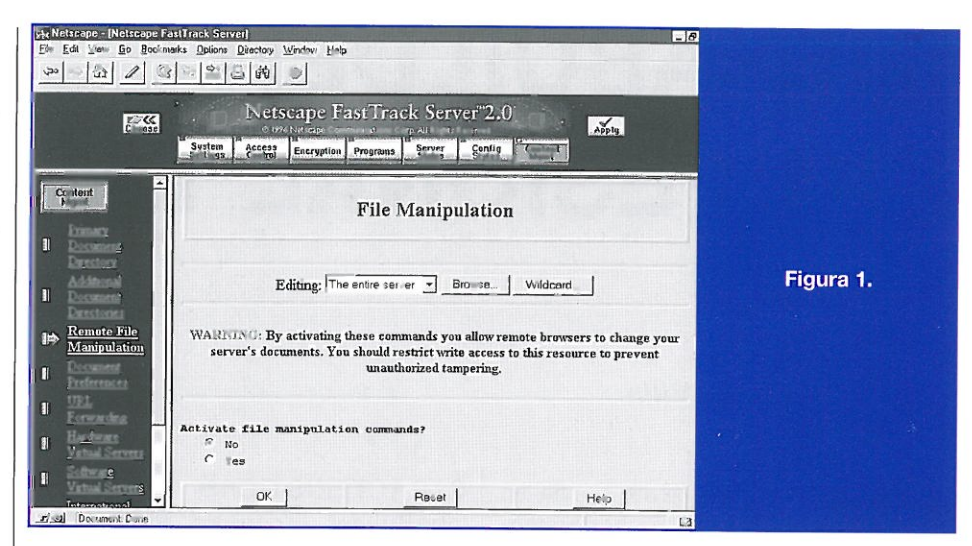


Figura 1.

- **IntraBuilder Professional:** Permite múltiples instancias del servidor IntraBuilder y proporciona soporte para los interfaces NSAPI, ISAPI y CGI. Incluye los servidores web Netscape FastTrack Web Server y Borland Web Server, además del motor de base de datos de Borland y los SQL Links para InterBase y Microsoft SQL Server.

- **IntraBuilder Client/Server:** Se trata de la edición más avanzada y está pensada para usuarios que trabajen con un gran volumen de información. Permite múltiples instancias del IDE y del servidor IntraBuilder en diferentes máquinas. También incluye el motor de Bases de datos de Borland y los SQL Links para los formatos de base de datos comentados anteriormente.

En el cuadro 1 se ofrece una comparativa de las características que ofrece cada una de las ediciones.

FUNCIONAMIENTO

El acceso a datos almacenados en una base de datos no se puede realizar directamente. Se necesitan una serie de niveles que organizan y controlan el flujo de la información desde el servidor de páginas web hasta la base de datos. IntraBuilder está estructurado en los siguientes niveles:

- **Servidor Web:** Proporciona la transferencia de páginas mediante el protocolo HTTP (*HyperText Transfer Protocol*). El navegador del usuario se conecta con este servidor para recibir la página web que ve en su pantalla.

- **IntraBuilder Broker:** Proporciona un interfaz de comunicación entre los agentes de IntraBuilder y el servidor web. Para ello utiliza los interfaces estándar CGI, ISAPI y NSAPI. Este elemento realiza un encaminamiento de peticiones inteligente. Es capaz de enviar las peticiones al agente IntraBuilder más indicado para responder a cada petición.

- **IntraBuilder Agents:** Los agentes son los ejes centrales de IntraBuilder y realizan diversas tareas como la generación automática de páginas web. También interpretan los formularios e informes, además de controlar el acceso a datos y el formato de los mismos. Pueden

Las aplicaciones creadas con IntraBuilder pueden ser utilizadas tanto en una red local como en Internet

existir varios agentes funcionando a la vez con el fin de aumentar la eficiencia.

- **Borland DataBase Engine (BDE):** Es el motor de bases de datos al que antes se ha hecho referencia. Realiza los accesos a las bases de datos mediante ODBC y los APIs nativos según las instrucciones proporcionadas por los agentes.
- **Datos:** Almacenados físicamente en cualquier formato de base de datos soportado por el BDE.

El Broker, los agentes y el BDE están integrados en el servidor IntraBuilder. Este servidor no es más que un ejecutable que se arranca para dar servicio a todas las peticiones de páginas con acceso a base de datos. Cuando se ejecuta, lanza el número de agentes especificado en la configuración. En la edición Cliente/Servidor se pueden lanzar servidores en diferentes máquinas que trabajan contra las mismas bases de datos. De este modo, se aumenta notablemente la eficiencia y se disminuye la carga de los servidores.

El acceso a las bases de datos se realiza mediante el Borland Database Engine

EL ENTORNO INTEGRADO

Debido a las características del producto, el entorno integrado es bastante sencillo, sobre todo si se compara con los entornos habituales que incluyen los grandes compiladores de lenguajes como C++ y Basic. La actividad de desarrollo se centra en el IntraBuilder Explorer. Se trata de un explorador de directorios basado en el de Windows 95 y Windows NT 4.0, pero con algunas diferencias. Está organizado en diferentes apartados: formularios, informes, *scripts*, tablas, consultas, imágenes, personalizado y todos. Cada uno de ellos muestra el tipo de archivos correspondiente. El directorio sobre el que trabaja el Explorer se puede cambiar con una caja *combo* situada en la parte superior que almacena los directorios visitados recientemente, o bien con un botón que da paso a una ventana desde la cual se puede elegir cualquier directorio del disco duro. Se utiliza este modo de trabajo porque, normalmente, las páginas web relacionadas con un mismo tema se almacenan en un mismo directorio al que tenga acceso el servidor web. En ese directorio también estarán los archivos relacionados con esas páginas, como pueden ser los gráficos y las tablas.

Además, el Explorer es cómodo de usar. En primer lugar, organiza de forma apropiada los distintos tipos de archivos. En segundo lugar, permite seleccionar, modificar y ejecutar algunos de los archivos directamente. Por ejemplo, para ejecutar un formulario basta con arrastrar el icono que lo representa fuera del Explorer y soltarle en el escritorio del entorno integrado. Además, presenta menús de contexto pulsando con el botón derecho del ratón sobre los objetos.

Todas las secciones menos personalizado (*custom*) muestran un icono con el nombre "untitled" (sin título). Haciendo doble click con el ratón sobre el mismo se crea un nuevo elemento. En el caso de los formularios, informes y tablas se pregunta al usuario si desea crear el elemento utilizando el experto o el diseñador correspondiente. Las consultas y las imágenes arrancan, respectivamente, el diseñador visual de consultas y el programa Paint de Windows.

DISEÑO DE TABLAS

El diseño de tablas en IntraBuilder es extremadamente sencillo. Cuando se crea una tabla nueva aparece un cuadro en el que se introducen los nombres de los campos y sus características (tipo de datos y tamaño, entre otras). Con la ayuda de los menús de contexto y la posibilidad de arrastrar y soltar se pueden insertar y eliminar campos fácilmente, además de poder intercambiar sus posiciones en la tabla. El formato de la tabla se puede elegir con una caja *combo* con las opciones disponibles. Si el formato soporta características adicionales, aparecerá el Inspector para poder introducir datos como el límite inferior y superior de un campo numérico, y si dicho campo puede dejarse vacío o no cuando se rellena un registro de la tabla. Una vez creada se puede cambiar la estructura de una tabla sin problemas. También se soporta la integridad referencial y opciones de seguridad.

DISEÑO DE FORMULARIOS

Los formularios, al fin y al cabo, son páginas web que permiten interactuar con una base de datos. Para construir un nuevo formulario no hay más que seleccionar los elementos necesarios

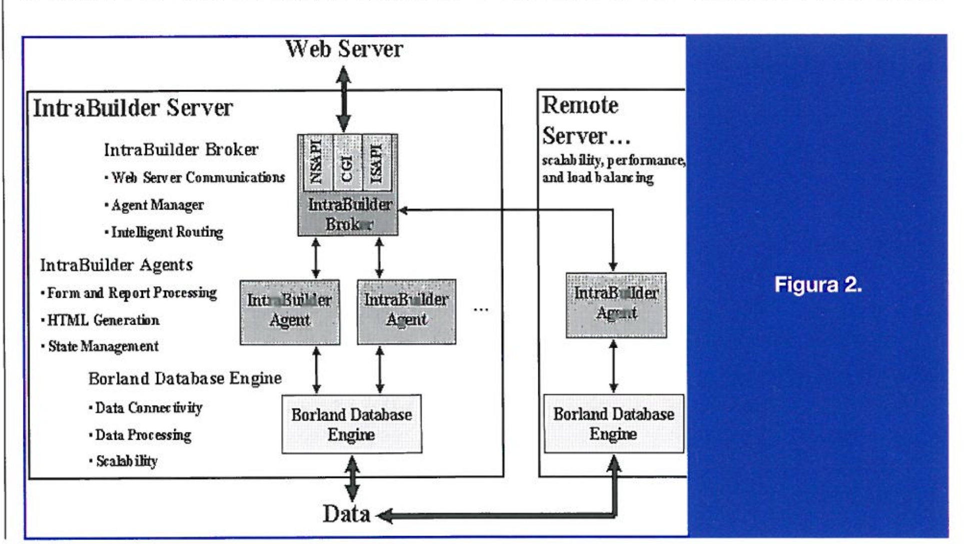
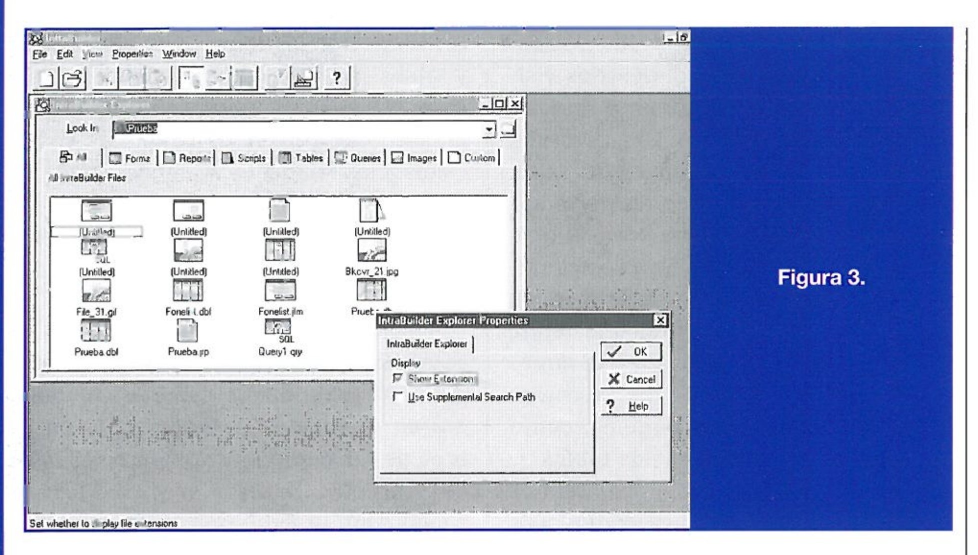


Figura 2.


Figura 3.

de la paleta de componentes. Dicha paleta está dividida en los apartados estándar y acceso a datos. La sección estándar contiene los componentes clásicos como los botones, cajas desplegables, imágenes y cajas de texto. Además, hay componentes especiales para las páginas web, como HTML, *applets* de Java y ActiveX. La otra sección incluye los componentes orientados al manejo de datos: consulta, base de datos y sesión. Ninguno de ellos es visible y se utilizan exclusivamente para asociar un formulario con una base de datos, poder realizar consultas y activar el bloqueo de registros para evitar inconsistencia de información.

Existen otras paletas, como la de alineación de componentes y la de campos. La última es especialmente útil debido a que contiene todos los campos de una base de datos, que pueden ser seleccionados para incorporarlos directamente en el formulario. Por supuesto, el enlace del campo situado en el formulario con el de la base de datos se hace de forma automática, con la comodidad que esto supone.

El aspecto del formulario se puede modificar fácilmente con los esquemas. Existen una serie de esquemas determinados que contienen un fondo para la ventana y un color y tipo de letra para los títulos y el texto que debe aparecer en la página web. Se pueden crear esquemas nuevos y utilizarlos posteriormente en nuevos formularios.

DISEÑO DE INFORMES

Los informes simplemente muestran una serie de datos en una página web. El diseñador de este tipo de documentos es muy similar al de informes, pero con menos posibilidades, puesto que no permite modificar la información.

El número de componentes que se pueden utilizar aquí es menor, aunque aparece una sección nueva en la paleta de componentes con un elemento adicional. Dicho elemento permite crear un grupo de campos cuya utilidad reside en mejorar la presentación del informe.

CREACIÓN DE CONSULTAS

Las consultas se crean mediante el *Visual Query Builder.* Esta utilidad, integrada en el entorno de desarrollo, permite realizar consultas a una base de datos utilizando el lenguaje SQL. El diseño de las consultas se hace de modo visual y no son necesarios conocimientos de SQL para utilizarlo.

CONSTRUCCIÓN DE PÁGINAS

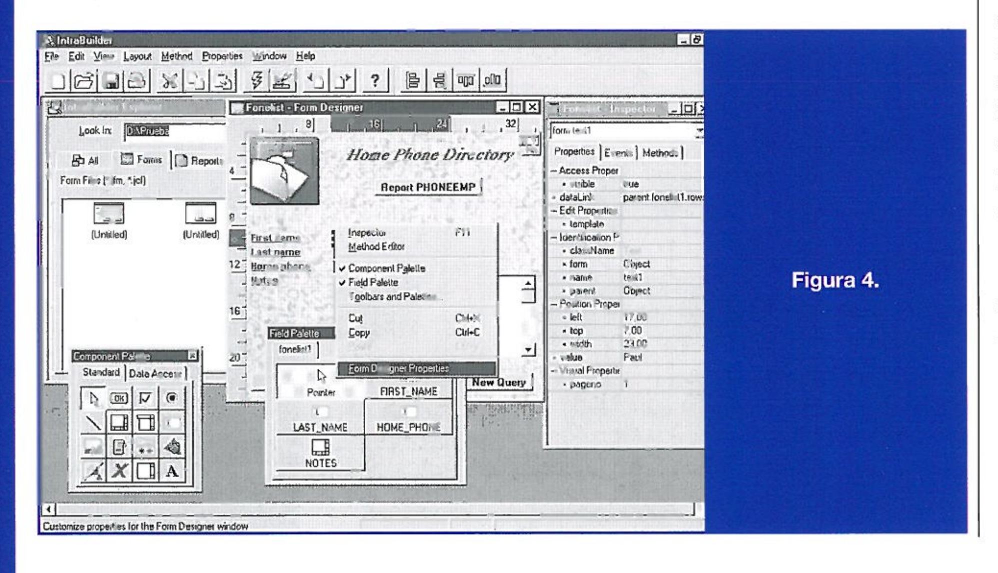
IntraBuilder incorpora un experto para la creación de páginas web principales (*home pages*). El experto es muy simple y crea una página básica que en la mayoría de los casos necesitará mejoras posteriores.

Para realizar estas mejoras se puede utilizar el Netscape Navigator Gold 3.0, incluido en todas las ediciones del producto. El navegador incluye un completo editor WYSIWYG (lo que se ve es lo que se obtiene) de páginas web.

COMENTARIOS FINALES

IntraBuilder es un producto completo, que proporciona todo lo que se espera de él. Dispone de una buena documentación y contiene herramientas de indudable calidad. Tanto su instalación como su configuración apenas presentan inconvenientes y se realizan de forma rápida.

En definitiva, una herramienta útil y necesaria que llega en el momento adecuado.


Figura 4.

CONCEPTOS DE PROGRAMACIÓN: LOS OBJETOS

Juan Manuel y Luis Martín

Los objetos pueden definirse como combinaciones de código y datos que pueden ser tratados como una sola unidad. Visual Basic puede trabajar con objetos como si se tratase de otro tipo de datos, permitiendo crear variables que contienen objetos para manipularlos. El origen de los objetos puede ser tanto el propio Visual Basic como otras aplicaciones externas. Para ello, la tecnología OLE permite estandarizar el acceso a los mismos. Son muchas las aplicaciones que admiten esta tecnología, como Microsoft Excel, Microsoft Word, etc.

A diferencia de versiones anteriores, la versión 4.0 de Visual Basic permite al programador crear sus propias clases, utilizando para ello los módulos de clase. A partir de estas clases es posible crear nuevos objetos. En este artículo se describen los aspectos relacionados con el manejo tanto de clases como de objetos.

OBJETOS, CLASES E INSTANCIAS

Los objetos, aunque son tratados como una sola unidad, constan de dos partes claramente diferenciadas:

- **Datos:** Son los atributos que definen las características propias del objeto. Generalmente, consisten en variables que contienen información relativa al objeto.

- **Acciones:** Son las operaciones que el objeto puede realizar. Generalmente, consisten en procedimientos que realizan una acción relacionada con el objeto.

En Visual Basic, a los datos de un objeto se les denomina *propiedades*, mientras que a las acciones que puede realizar se les denomina *métodos*. Por tanto, cada uno de los objetos manejados por Visual Basic tendrá su propio repertorio de propiedades y métodos.

Los objetos de un mismo tipo se crean a partir de una definición general denominada *clase*. Las clases pueden considerarse como el "molde" a partir del cual se crean los objetos de un determinado tipo. Cada uno de los objetos creados a partir de una clase se denomina *instancia*. Así, los controles de la caja de herramientas representan las distintas clases de los objetos control. Cuando se añade un nuevo control al formulario a partir de una de esas clases, se está creando una nueva instancia del objeto.

Cuando se trabaja con un formulario en tiempo de diseño se está creando una nueva clase. En tiempo de ejecución se creará una instancia del formulario a partir de esa clase. A diferencia de los controles, cada uno de los formularios creados en fase de dise–o representa una clase distinta. En general, puede considerarse que un módulo de formulario es un módulo de clase con interfaz visible.

Inicialmente, las instancias que se van creando a partir de una misma clase son iguales. Posteriormente, es posible asignarles distintas propiedades, incluido un nuevo nombre, con el fin tanto de diferenciarlas entre sí como de dotarlas de características propias.

Los objetos manejados por Visual Basic pueden ser tanto propios como procedentes de otras aplicaciones y herramientas de desarrollo. Con Visual Basic es posible manipular objetos externos tales como un gráfico procedente de Microsoft Paint, una hoja de

Las aplicaciones desarrolladas con Visual Basic trabajan con unas entidades denominadas objetos. En artículos anteriores, los únicos objetos que han sido manejados son los controles y formularios. Sin embargo, una base de datos, una hoja de cálculo o un gráfico también pueden ser objetos.

cálculo de Microsoft Excel o un documento de Microsoft Word. Este tipo de intercambio de objetos entre distintas aplicaciones es posible gracias a que éstas son compatibles con un estándar denominado OLE, Vinculación e Incrustación de Objetos (*Object Linking and Embedding*).

A las aplicaciones que proporcionan objetos a otras aplicaciones se les denomina servidores OLE. La forma de acceder a sus objetos e interfaces se define en la librería de tipos de la aplicación. Igualmente, a las aplicaciones que hacen uso de objetos de otras aplicaciones se les denomina clientes OLE.

Desde el código de Visual Basic es posible manejar los objetos externos accediendo a sus propiedades y métodos. Si el objeto tiene un interfaz visible por el usuario, se denomina objeto insertable. Los objetos insertables pueden ser a–adidos a un formulario utilizando el control *Contenedor OLE*, o bien incluyéndolos en la caja de herramientas con la ventana de controles personalizados.

Sin embargo, Visual Basic también permite utilizar objetos que no sean visibles por el usuario, ya sea por carecer de interfaz visible o simplemente porque no se desea visualizarlos. En este caso, es necesario a–adir una referencia a la librería de objetos de procedencia, lo que permitirá crear instancias del objeto.

UTILIZACIÓN DE OBJETOS

Como ya se vio en artículos anteriores, el manejo con Visual Basic de las propiedades en tiempo de ejecución se realiza de forma similar al de las variables. Para asignar valores a una propiedad de un objeto es necesario indicar el nombre del objeto y el de la propiedad, separados por un punto.

Objeto.Propiedad = Expresión

Igualmente, es posible obtener el valor de una propiedad asignando su valor a una variable.

Variable = Objeto.Propiedad

En general, el valor de una propiedad se obtiene incluyendo ésta en una expresión, teniendo en cuenta el tipo de datos tanto de la propiedad como del resto de operandos de la expresión.

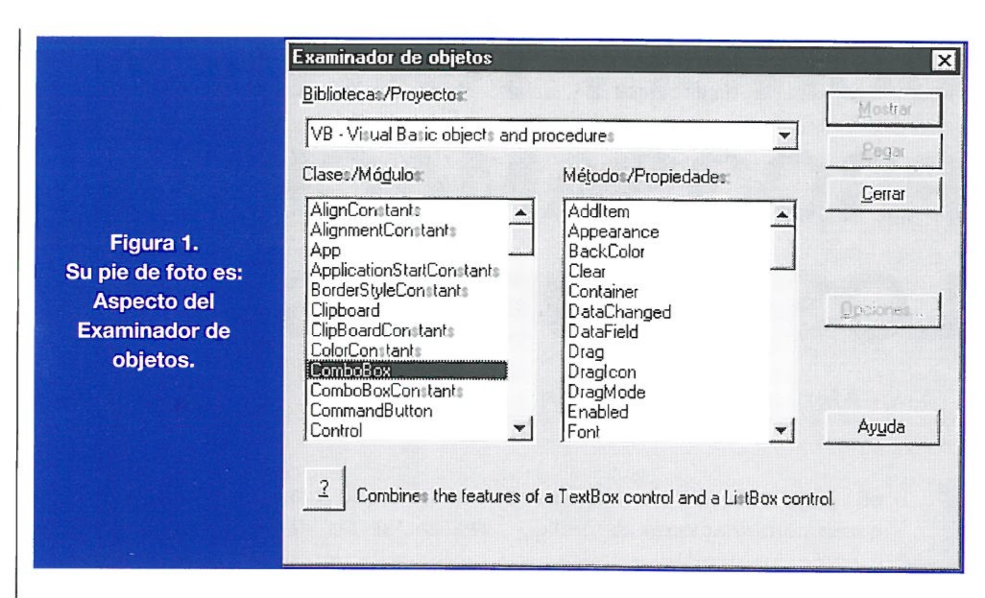


Figura 1. Su pie de foto es: Aspecto del Examinador de objetos.

La sintaxis para la llamada a un método cuando éste no devuelve ningún valor, es decir, cuando se trata de una subrutina, puede expresarse de la forma siguiente:

Objeto.Método [ListaArgumentos]

En el caso de que el método devuelva algún valor, es decir, cuando se trata de una función, será necesario encerrar los argumentos entre paréntesis. En este caso, el valor devuelto puede asignarse a una variable o, en general, ser utilizado dentro de una expresión.

Variable = Objeto.Método ([ListaArgumentos])

EL EXAMINADOR DE OBJETOS

Visual Basic dispone de un cuadro de diálogo que permite visualizar las características de los objetos disponibles para ser utilizados en el proyecto. Se trata del *Examinador de objetos*, que muestra las clases disponibles tanto en las librerías de objetos como en los distintos módulos que forman parte del proyecto, mostrando también sus propiedades y métodos. Para acceder al él, debe seleccionarse la opción *Examinador de objetos* del menú *Ver*, hacer clic sobre el botón *Examinador de objetos* de la barra de herramientas o pulsar *[F2]*. La figura 1 muestra el aspecto del examinador de objetos.

El examinador de objetos se compone de los siguientes elementos:

- *Bibliotecas/Proyectos*: Permite seleccionar entre las distintas librerías de objetos disponibles para la aplicación, ya sean procedentes de Visual Basic o de otras aplicaciones externas. Entre ellas, se incluye el propio proyecto actual.
- *Clases/Módulos*: Muestra los nombres de todas las clases disponibles en la librería.
- *Metodos/Propiedades*: Muestra las propiedades y los métodos de la clase. Algunas librerías incluyen también constantes intrínsecas.
- *?*: Muestra la ayuda interactiva correspondiente al elemento seleccionado en las listas anteriores.
- *Pegar*: Inserta en la ventana de código el elemento actualmente seleccionado en *Propiedades/ Métodos*.
- *Mostrar*: Permite situarse en el procedimiento del proyecto actualmente seleccionado del proyecto.
- *Opciones*: Abre un nuevo cuadro de diálogo, denominado *Opciones de miembro*, que permite incluir información adicional referente al elemento seleccionado.

VARIABLES OBJETO

Las variables también pueden utilizarse para almacenar referencias a objetos, como si éstos fuesen datos de alguno

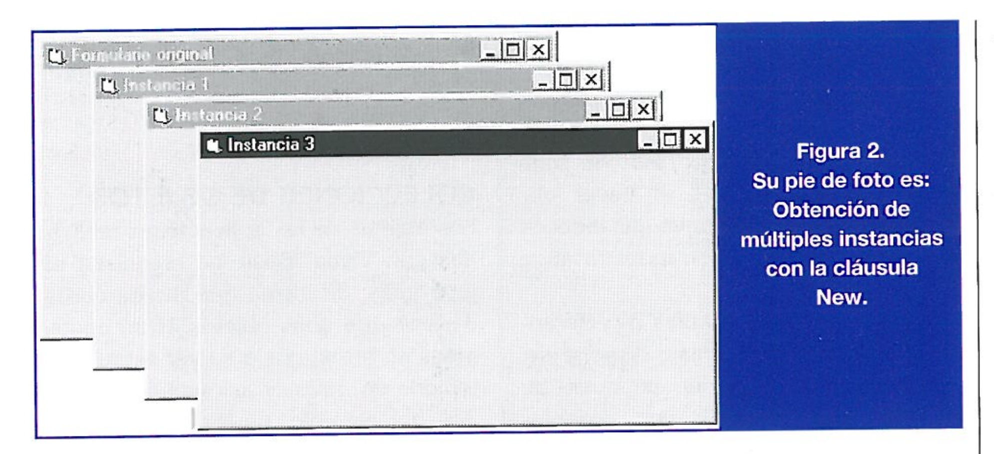


Figura 2. Su pie de foto es: Obtención de múltiples instancias con la cláusula New.

de dichos tipos. A este tipo de variables se les denomina *variables objeto*. La utilización de variables para hacer referencia a objetos presenta las mismas ventajas que cuando se utilizan para manejar cualquiera de los otros tipos de datos. Así, por ejemplo, es posible ir cambiando el valor contenido en una misma variable objeto, de forma que ésta vaya haciendo referencia a distintos objetos.

La declaración de variables objeto se realiza de forma similar al resto de variables.

{Dim | ReDim | Static | Private | Public} Variable As [New] Clase

El tipo de datos de la variable se indica en el argumento *Clase* que, como su nombre indica, será el nombre de una clase. Una vez declarada la variable objeto, ésta sólo podrá hacer referencia a objetos pertenecientes a la clase correspondiente.

Es posible declarar variables objeto que hagan referencia a un tipo de control específico, como *TextBox*, *ListBox*, *Label*, etc... Igualmente, es posible declarar variables objeto que hagan referencia a un tipo específico de formulario de la aplicación. A este tipo de variables de les denomina variables objeto específicas.

Dim Tb As TextBox íDeclara una variable de tipo TextBox
Dim F1 As New Form1 íDeclara una variable de tipo Form1

En los ejemplos anteriores, la variable *Tb* puede hacer referencia a cualquier cuadro de texto de la aplicación. Por su parte, la variable *F1* sólo podrá hacer referencia a instancias del formulario *Form1*.

Sin embargo, también es posible declarar variables objeto que hagan referencia a cualquier tipo de control, a cualquier tipo de formulario e incluso a cualquier tipo de objeto. A este tipo de variables se les denomina variables objeto genéricas. Visual Basic dispone de cuatro tipos de objetos genéricos, que permiten declarar variables para hacer referencia a distintos tipos de objetos:

- ***Form***: Puede hacer referencia a cualquier formulario de la aplicación, incluido el formulario MDI.
- ***Control***: Puede hacer referencia a cualquier tipo de control disponible en la aplicación.
- ***MDIForm***: Sólo puede hacer referencia al formulario MDI.
- ***Object***: Puede hacer referencia a cualquier objeto de la aplicación, ya sea un formulario o un control.

Visual Basic permite declarar variables objeto que hagan referencia a un formulario específico de la aplicación, haciendo uso de su nombre como nombre de la clase. Sin embargo, no es posible declarar variables objeto que hagan referencia a un control específico.

Dim Cnt As Control íReferencia a cualquier control
Dim Frm As Form íReferencia a cualquier formulario
Dim Obj As Object íReferencia a cualquier objeto

Dim Tb As Text1 íDeclaración incorrecta

Para que una variable objeto haga referencia a un objeto determinado, éste debe ser asignado a la variable. Esta operación se realiza mediante la sentencia *Set*, cuya sintaxis es la siguiente:

Set Variable = {[New] Objeto | Nothing}

En *Objeto* debe indicarse el nombre del objeto que se desea asignar, y debe ser del mismo tipo de objeto con el que se declaró la variable. En caso contrario, se producirá un error. También es posible utilizar otra variable objeto del mismo tipo, o un objeto devuelto por un método o función. Por tanto, puede decirse que el argumento *Objeto* representa una expresión de tipo objeto.

Para eliminar la referencia de una variable objeto a cualquier objeto existente debe asignársele el valor especial *Nothing*. Cuando se asigna este valor, la variable no hace referencia a ningún objeto.

Dim Tb As TextBox íDeclarar variable de tipo TextBox
Set Tb = Text1 íAsignar a la variable un cuadro de texto
Tb.Text = "Hola" íCambiar propiedad mediante la variable
Set Tb = Nothing íDesasignar la variable

Las variables objeto pueden ser asignadas entre sí. De esta forma, es posible hacer que una variable objeto haga referencia al mismo objeto que otra, simplemente asignándola a ésta.

Set Obj1 = Label1 íObj1 hace referencia a Label1
Set Obj2 = Obj1 íObj2 también hace referencia a Label1

Las variables objeto también pueden ser utilizadas como parámetros en las llamadas a procedimientos. De esta forma, es posible pasar un objeto al procedimiento sin más que especificarlo en la lista de argumentos. El parámetro correspondiente será asignado de tal forma que hará referencia al objeto pasado, permitiendo su manipulación desde dentro del procedimiento. Así, por

ejemplo, el siguiente procedimiento reduce a la mitad el tama–o de un control.

```
Private Sub Mitad(Ctrl As Control)
  Ctrl.Width = Ctrl.Width / 2
  Ctrl.Height = Ctrl.Height / 2
End Sub
```

La cláusula *New* permite crear nuevas instancias de una determinada clase en tiempo de ejecución. Para ello, debe ser incluida en la declaración de una variable objeto o en una sentencia de asignación de la misma. Esta cláusula sólo puede ser utilizada con formularios específicos de la aplicación, clases definidas en los módulos de clase, colecciones de objetos y objetos de automatización OLE.

Cuando se utiliza la cláusula *New* en una sentencia de asignación (*Set*), se crea automáticamente una nueva instancia de la clase, que será asignada a la variable especificada. Así, por ejemplo, para crear una nueva instancia del formulario por defecto y visualizarla, puede utilizarse lo siguiente:

```
Dim Frm As Form1
Set Frm = New Form1  íCrea la nueva instancia
Frm.Show íMuestra la instancia creada
```

Sin embargo, también es posible incluir la cláusula *New* en la sentencia de declaración de la variable objeto. En este caso, no será necesario utilizar la sentencia de asignación, aunque la instancia no se creará hasta que se haga referencia a la variable.

```
Dim Frm As New Form1
Frm.Show íCrea y muestra la nueva instancia
```

Cuando se crean instancias de un formulario, éstas no se visualizan inicialmente, ya que tienen su propiedad *Visible* a *False*. Además, en el caso de los formularios no basta con asignar *Nothing* a la variable o dejarla fuera de alcance para eliminar el formulario, sino que debe descargarse explícitamente con la sentencia *Unload*.

La cláusula *New* no puede ser utilizada con variables objeto de tipo genérico, ni con variables objeto de tipo control específico. El cuadro 1 contiene un ejemplo que muestra cómo pueden crearse múltiples instancias de un formulario con la cláusula *New*. Se trata de un formulario que, al hacer clic sobre él, genera una nueva instancia de sí mismo, tal y como se muestra en la figura 2.

Para comparar variables de referencia a objetos, Visual Basic dispone del operador *Is*. Se trata de un operador especial que compara dos variables objeto, devolviendo *True* cuando ambas hacen referencia al mismo objeto y *False* en caso contrario.

```
Set Obj1 = Text1
Set Obj2 = Text1
Set Obj3 = Text2
Print Obj1 Is Obj2 íResultado: True
Print Obj2 Is Obj3 íResultado: False
```

Este operador también puede ser utilizado en combinación con la palabra reservada *TypeOf*, con el fin de realizar comprobaciones del tipo de objeto.

```
TypeOf VarObjeto Is TipoObjeto
```

En el *VarObjeto* debe especificarse el nombre de una variable o referencia a un objeto, mientras que en *TipoObjeto* debe especificarse cualquiera de los tipos de objetos de la aplicación, ya sea genérico o específico. Este tipo de construcción sólo puede ser utilizado como condición de una sentencia de control *If...Then...Else*.

```
If TypeOf Obj Is TextBox Then
  Print "Se refiere a un cuadro de texto"
ElseIf TypeOf Obj Is Label Then
  Print "Se refiere a una etiqueta"
ElseIf TypeOf Obj Is CommandButton
  Print "Se refiere a un botón de comando"
End If
```

COLECCIONES DE OBJETOS

Los objetos de las aplicaciones realizadas con Visual Basic se organizan en jerarquías, de forma que puede considerarse que unos objetos están contenidos en otros, que a su vez están contenidos en otros, y así sucesivamente. Así, por ejemplo, un formulario puede contener varios controles, que a su vez contienen otros controles. La figura 3 muestra un ejemplo esquemático de este tipo de organización.

Como ya se ha mencionado anteriormente, para hacer referencia en el código a un objeto que se encuentra contenido en otro, es necesario indicar el nombre de ambos, separados por un punto.

```
Form1.Text1.Font.Name = "LinePrinter"
```

Visual Basic dispone de un tipo especial de objetos que permite realizar agrupaciones de otros objetos. Se trata de los objetos colección (*Collection*), que están constituidos a su vez por un cierto número de objetos ordenados. Cada uno de los objetos de la colección se denomina miembro. La única relación que existe entre los miembros de una determinada colección es que pertenecen a ésta, no siendo necesario que sean del mismo tipo.

La creación de un objeto colección se realiza de forma similar a la de cualquier otro objeto, utilizando las mismas sentencias, aunque especificando como tipo de datos la palabra reservada *Collection*.

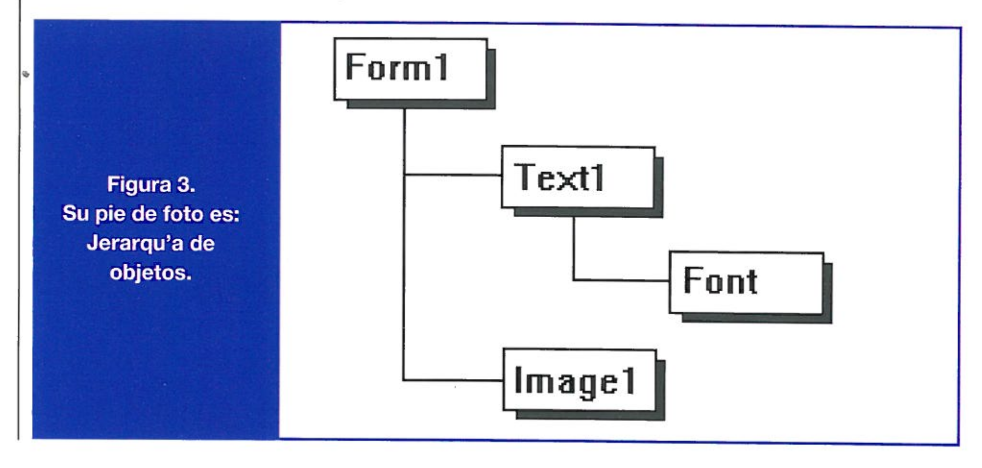


Figura 3. Su pie de foto es: Jerarqu'a de objetos.

Dim Grupo As New Collection

Dentro del objeto colección, los miembros se sitúan ordenadamente, de forma similar a como lo hacen los elementos de una matriz. Así, a cada objeto le corresponde un número de índice que indica su posición dentro de la colección. De esta forma, es posible hacer referencia a un miembro indicando el nombre de la colección, seguido del valor del índice encerrado entre paréntesis. También es posible hacer referencia a un miembro especificando una clave de identificación en lugar del índice. Esta clave se indica cuando se añade el miembro a la colección, y debe expresarse como una cadena de caracteres. Finalmente, es posible indicar la clave del miembro literalmente, separándola del nombre de la colección mediante el carácter admiración (!).

Así, por ejemplo, si el segundo elemento de la colección *Grupo* es un cuadro de texto cuya clave de identificación es *T2*, es posible acceder a su propiedad *Text* con los siguientes tipos de sintaxis:

```
Grupo(2).Text = "Hola"
Grupo("T2").Text = "Hola"
Grupo!T2.Text = "Hola"
```

Los objetos colección disponen sólo de la propiedad *Count*, que devuelve el número total de miembros que contiene la colección.

```
For N = 1 To Grupo.Count
  ¡Operación con el miembro N-ésimo
Next
```

Una vez creado un objeto colección, es posible añadirle nuevos miembros utilizando para ello el método *Add*.

```
Objeto.Add(Miembro[, Clave[, Antes[, Después]]])
```

En el argumento *Miembro* se indica el nombre del objeto que se desea agregar a la colección. Por su parte, el argumento *Clave* permite especificar una cadena de caracteres que identificará al miembro cuando se desee acceder a él sin utilizar el índice posicional.

```
Dim Grupo As New Collection
Grupo.Add Text2, "T2"
```

Por defecto, cada vez que se a–ade un nuevo miembro a una colección, éste se sitúa al final de la misma. Sin embargo, es posible especificar la posición donde se desea insertar el miembro, utilizando para ello los argumentos *Antes* y *Después*. Estos argumentos son mutuamente excluyentes, es decir, si se especifica uno de ellos no puede especificarse el otro. En *Antes* puede especificarse una posición entre 1 y *Count*. De esta forma, el nuevo miembro se situará delante del miembro indicado. También es posible indicar una clave de identificación en vez del índice posicional. El argumento *Después* es muy similar, aunque el nuevo miembro será insertado detrás del miembro especificado.

```
Grupo.Add (Text1,"T1","T2")          ¡ T 1
se inserta delante de T2
```

Para eliminar un miembro de una colección se utiliza el método *Remove*. La sintaxis de este método es la siguiente:

```
Objeto.Remove(Índice)
```

En el argumento *Índice* debe especificarse la posición del miembro que se desea eliminar, entre 1 y *Count*. También es posible indicar la clave de identificación del miembro a eliminar.

```
Grupo.Remove ("T1") ¡Elimina el miembro T1

Grupo.Remove (Grupo.Count) ¡Elimina el último miembro
```

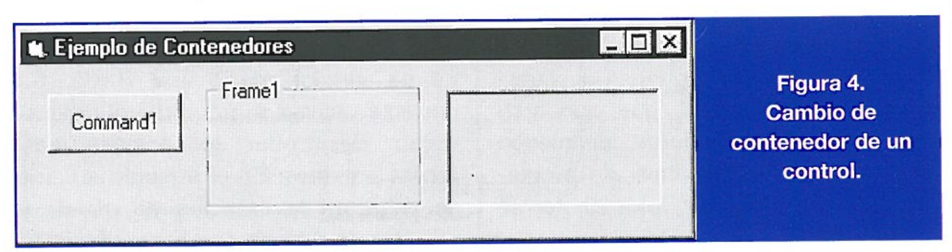


Figura 4. Cambio de contenedor de un control.

La sentencia *For Each...Next* puede ser utilizada para recorrer todos los miembros de una colección. Así, por ejemplo, para imprimir la propiedad *Caption* de una colección de etiquetas puede utilizarse el siguiente fragmento de código:

```
For Each Elemento In Etiquetas
  Print Elemento.Caption
Next
```

Visual Basic dispone de varias colecciones de objetos predefinidas, que permiten manipular los elementos más utilizados. De esta forma, es posible manejar los formularios de una aplicación, los controles de un formulario o las impresoras disponibles del sistema mediante el uso de una colección de objetos.

La colección *Forms* contiene todos los formularios de la aplicación que se encuentran cargados. Así, por ejemplo, para poner en rojo el color de fondo de los formularios cargados puede utilizarse el siguiente fragmento de código:

```
For Each Frm In Forms
  Frm.BackColor = vbRed
Next
```

La colección *Controls* contiene todos los controles de un formulario. Así, por ejemplo, para deshabilitar los botones de comando de un formulario puede utilizarse el siguiente código:

```
For Each Ctrl In Controls
  If TypeOf Ctrl Is CommandButton Then
Ctrl.Enabled = False
Next
```

Por último, la colección *Printers* contiene todas las impresoras disponibles del sistema. Así, por ejemplo, para establecer como impresora predeterminada la primera impresora capaz de imprimir en color puede utilizarse el siguiente fragmento de código:

```
For Each Prn In Printers
  If Prn.ColorMode = vbPRCMColor Then
    Set Printer = Prn
    Exit For
  End If
Next
```

La utilización de controles contenedores también constituye una jerarquía de objetos. Dentro de los controles estándar, los contenedores son los marcos y cuadros de dibujo, además de los propios formularios. Así, por ejemplo, es posible situar un botón de comando en un cuadro de dibujo que se encuentra dentro de un marco que, a su vez, se encuentra dentro del formulario.

En tiempo de ejecución es posible acceder al contenedor de un control a través de su propiedad *Container*, que representa el objeto contenedor. También es posible cambiarlo de contenedor, asignando un nuevo control a la propiedad *Container* con la sentencia *Set*. El cuadro 2 contiene un ejemplo que muestra cómo un control puede ser cambiado de contenedor en tiempo de ejecución, accediendo a su propiedad *Container*.

CREACIÓN DE CLASES PROPIAS

A diferencia de versiones anteriores, la versión 4.0 de Visual Basic permite al programador crear sus propias clases. Para ello, se utiliza un nuevo tipo de módulos, denominados módulos de clase (*Class module*). Se trata de módulos muy similares a los de formulario. La principal diferencia entre ambos es que los módulos de formulario presentan un interfaz visible, mientras que los de clase no.

Cada módulo de clase incluido en el proyecto sólo permite definir una única clase. Las propiedades de dicha clase deben ser definidas mediante la utilización de variables a nivel de módulo o procedimientos de propiedades. Los métodos de la clase deben ser definidos mediante procedimientos *Sub* y *Function*, según devuelvan o no un valor. Los objetos o instancias sólo podrán ser creados a partir de la clase en tiempo de ejecución.

Para crear un nuevo módulo de clase debe seleccionarse la opción *Módulo de Clase* del menú *Insertar*. Según se van añadiendo nuevos módulos de clase, Visual Basic les asigna los nombres *Clase1*, *Clase2*, etc... Cada módulo de clase dispone de tres propiedades predefinidas:

- *Name*: Contiene el nombre de la clase en forma de cadena de caracteres. Inicialmente coincide con el nombre del módulo.
- *Public*: Contiene un valor lógico que indica si la clase será accesible desde otras aplicaciones fuera del proyecto.
- *Instancing*: Contiene un valor numérico que indica la forma en que puede accederse a la clase desde otras aplicaciones cuando la propiedad *Public* se encuentra a *True*.

La propiedad *Name* determina el nombre de la clase, y su valor será el que se utilice a la hora de declarar objetos de dicha clase. Para ello se utilizan las mismas sentencias que en la creación de instancias de formularios, incluyendo también la palabra reservada *New*.

Dim Variable As New Clase

Una vez creado un nuevo módulo de clase en el proyecto, es posible añadir propiedades al mismo. La forma mas sencilla consiste en declarar variables públicas a nivel de módulo. La declaración de las propiedades con la sentencia *Public* permitirá acceder a éstas desde otros módulos cuando se haya creado una instancia de la clase. Esto también incluye a la propiedad *Name* del objeto, que aunque ya se encuentra definida, debe ser declarada siempre que vaya a ser utilizada desde otro módulo.

Public Name As String

También es posible añadir nuevas propiedades a la clase mediante la creación de procedimientos de propiedades. La utilización de este tipo de procedimientos en vez de variables públicas en un módulo de clase permite crear propiedades de sólo lectura para dicha clase. Para ello pueden declararse variables privadas, que quedarán ocultas a otros módulos, definiendo también un procedimiento de propiedad que devuelva su valor de forma controlada.

La creación de métodos en un módulo de clase se realiza mediante la definición de subrutinas y funciones públicas. Éstas podrán ser utilizados con los objetos creados en tiempo de ejecución, y permitirán acceder desde otros módulos a las propiedades de la clase que se encuentran ocultas. Los métodos que devuelven valores deben ser definidos como funciones, mientras que los que no devuelven ningún valor pueden ser definidos como subrutinas. Además, es posible definir métodos privados que sólo podrán ser llamados desde los procedimientos del propio módulo de clase.

Los módulos de clase sólo disponen de dos procedimientos de evento y, a diferencia de propiedades y métodos, no es posible a–adir nuevos procedimientos de evento al módulo. Se trata de los eventos *Initialize* y *Terminate*, que se producen cuando se crea o destruye una nueva instancia de la clase, respectivamente.

El evento *Initialize* se produce cuando se crea un objeto a partir de la clase. Para ello, bastará con hacer referencia a alguna propiedad del mismo, después de haber sido declarado o asignado a una variable con la palabra clave *New*. Por su parte, el evento *Terminate* se produce cuando se destruye el objeto, es decir, cuando todas las variables que hacen referencia a él se ponen a *Nothing* o quedan fuera del ámbito de visibilidad.

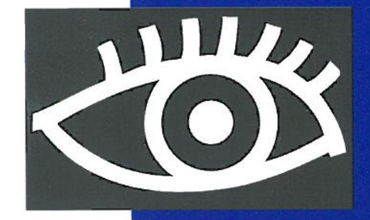
El cuadro 3 muestra un ejemplo que ilustra el funcionamiento de estos dos eventos. Cada vez que se crea o destruye un objeto de la clase *Prueba*, se visualiza un mensaje indicándolo. Para terminar la aplicación debe cerrarse el formulario. Si se termina utilizando el botón *Terminar* de la barra de herramientas, la ejecución se detiene inmediatamente, sin llegar a producirse el evento *Terminate* del objeto antes de descargarse el formulario.

CONCLUSIÓN

La capacidad para el manejo de objetos es una de las importantes mejoras de la última versión de Visual Basic. Su correcta utilización permitirá al programador desarrollar aplicaciones realmente potentes. En el próximo artículo se analizará la creación de menús y cuadros de diálogo con Visual Basic.

BITMAPS INDEPENDIENTES DE DISPOSITIVO

Jorge R. Regidor



Este mes se va a describir el proceso para cargar, salvar y visualizar con imágenes independientes de dispositivo (DIB). Las clases MFC 4.0 no contienen una clase específica que encapsule esta funcionalidad, pero dentro de los programas de ejemplo sí existe un programa que muestra la forma de realizar estas operaciones. Es este programa la mejor guía de cómo trabajar con los DIBs, y en este artículo se va a hacer un seguimiento de toda la arquitectura del programa, centrándonos más en lo que a DIBs se refiere.

Aunque, como el lector puede suponer, existen controles OCX o ActiveX que se encargan de encapsular la funcionalidad para mostrar imágenes independientes de dispositivo. Pero en este artículo se va a bajar hasta el nivel del API Win32 para realizar el programa. Para ello, se va a describir en detalle el programa ejemplo *DIBLook*, incluido dentro de los programas ejemplo de Visual C++ 4.0.

INSTALACIÓN DEL EJEMPLO

Para instalar los ficheros necesarios del ejemplo *DIBLOOK* se debe pulsar el botón de búsqueda o pulsar *CTRL+F*. Una vez que ha salido la ventana de búsqueda se inserta la cadena *DIBLOOK sample*. Una vez seleccionado, se pulsa *DISPLAY* y en ese momento aparecerá la ventana de ayuda con el ejemplo. En este punto, si se pulsa el botón, se copiarán los ficheros necesarios para poder trabajar con ellos.

Todos los ficheros de instalación se guardan en el directorio *C:\msdev\samples\mfc\general\diblook*, suponiendo que Visual C++ 4.0 está instalado en *C:\msdev*. Para abrir el proyecto, se pulsa *Open Workspace...* del menú *File* y se selecciona el fichero *diblook.mdb*.

DESCRIPCIÓN DE LA APLICACIÓN Y SUS FICHEROS

Este programa ejemplo es una aplicación MDI, donde en cada una de la ventanas hijas muestran la imagen cargada desde un fichero DIB o bien copiándola desde portapapeles. A partir de este momento se supone que el lector tiene accesibles los ficheros fuente de la aplicación *DIBLOOK*.

El proyecto *DIBLOOK* contiene los ficheros fuente que aparecen reflejados en la tabla adjunta. (Ver tabla)

mainfrm.cpp	Contiene la clase encargada de gestionar la ventana principal de la aplicación.
Diblook.cpp	Fichero que contiene la clase que gestiona la aplicación.
Dibview.cpp	Fichero que contiene la clase que gestiona las vistas de la aplicación, es decir, donde se van a mostrar las imágenes.
Dibdoc.cpp	Fichero que contiene la clase que gestiona los documentos para cargar y grabar las imágenes desde o hacia disco.
Dibapi.cpp	Fichero que contiene una serie de funciones para manejar los bitmaps independientes de dispositivo.
Myfile.cpp	Fichero que contiene dos funciones para cargar y guardar los ficheros del tipo DIB.

Ficheros fuente del proyecto Diblock.

Cabe destacar que en este ejemplo se trata también sobre BITMAPS de la versión 1.x, que por motivos obvios en el tiempo, se van a dejar de lado, centrándonos en la versión actual.

La visualización de imágenes de alta calidad en resolución y colores dentro de las aplicaciones es una de las partes más vistosas de la programación para Windows 95.

CARGAR UN DIB DESDE FICHERO

Cuando en la aplicación se pulsa sobre la opción *Abrir* del menú *Fichero*, la función *CDibDoc::OnOpenDocument* es llamada, lo que se traduce en una serie de pasos para cargar el DIB del fichero indicado.

La primera acción a emprender es abrir el fichero. Para ello, el entorno pasa en la función *CDibDoc::OnOpenDocument* el nombre del fichero a abrir como parámetro. Para abrir el fichero, se crean dos objetos locales a la función, uno del tipo *CFile* y otro del tipo *CFileException*. *CFile* es una clase que nos va a permitir abrir el fichero y acceder a sus datos. Para abrir el fichero se llama a la función miembro *Open* de la clase *CFile*, pasándole como primer parámetro el nombre del fichero, como segundo parámetro el modo de apertura y como tercer parámetro la dirección al objeto del tipo *CFileException*, que nos va a permitir conocer cualquier incidencia a la hora de abrir el fichero.

Una vez abierto de forma correcta el fichero, se llama a la función *ReadDIBFile*, incluida dentro del fichero *myfile.cpp*. Dentro de esta función se van a leer todas las estructuras necesarias para visualizar la imagen. La primera lectura, mediante la función miembro *Read* de la clase *CFile*, es una estructura del tipo *BITMAPFILEHEADER*. Uno de los campos de esta estructura es *bfType*, que debe de coincidir con los bytes *BM*, que indican que este fichero es un bitmap. De no ser así, podemos afirmar que el fichero abierto no es un bitmap.

Una vez leída esta estructura se debe reservar memoria para almacenar la información restante. Para ello, y debido al normalmente elevado tamaño a reservar, se utilizan funciones para reservar memoria global. El paso posterior es fijar el bloque en memoria para leer con la función *ReadHuge* desde disco. Una vez leído el bloque de disco, se debe desbloquear la memoria con *GlobalUnlock* y devolver el *handle* del bloque de memoria.

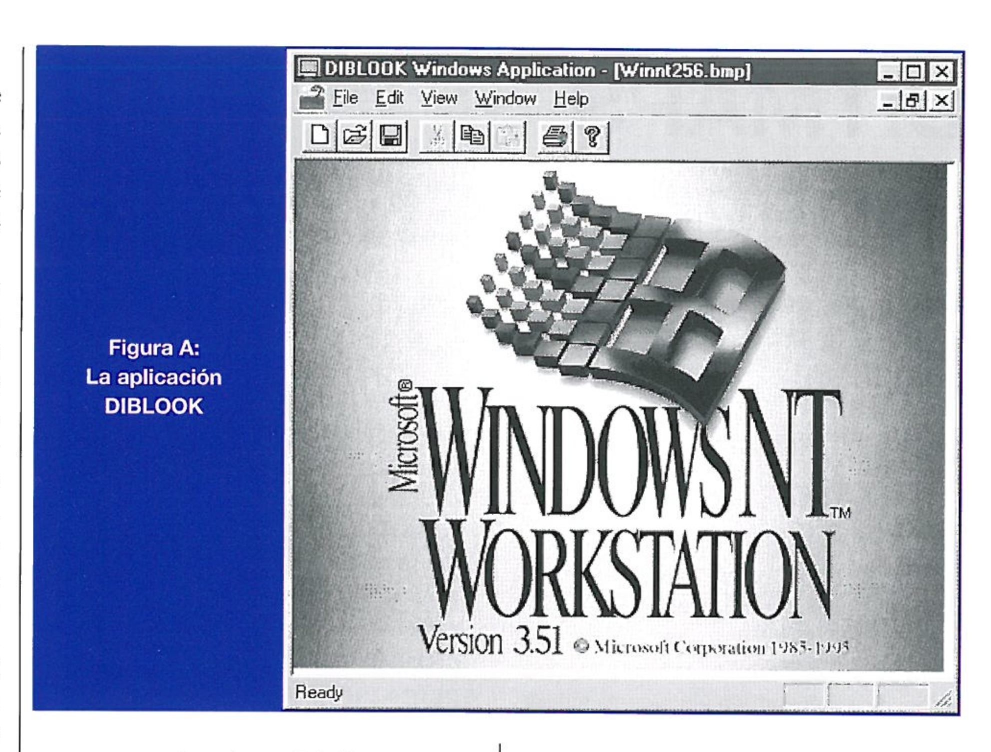


Figura A: La aplicación DIBLOOK

GESTIÓN DE MEMORIA GLOBAL

A modo de paréntesis, y sin profundizar más de lo necesario, conviene saber que cuando la cantidad de memoria a reservar es elevada, y debido a que tradicionalmente en Windows 3.x se ha reservado dentro de la memoria global, se utilizan las funciones del API para reservar memoria dentro del *heap* global del programa. La realidad es que en Win32, al ser un entorno de direccionamiento lineal de 32 bits, no existe diferencia entre reservar memoria dentro del heap local o global. Esto implica tener que usar una serie de funciones diferentes de los ya conocidos operadores *new* y *delete* de C++. Frente a estos operadores, se usan principalmente las siguientes funciones del API Win32.

HGLOBAL GlobalAlloc(UINT uFlags, DWORD dwBytes);

Esta función reserva un bloque de memoria de un tamaño *dwBytes*. Además, y en función del campo *uFlags*, puede inicializarlo de distintas maneras (Ver Tabla 1). Esta función devuelve el *handle* al bloque de memoria, no directamente su dirección, lo que permite en caso de necesidad descargar el bloque a disco.

LPVOID GlobalLock(HGLOBAL hMem);

Esta función permite bloquear la memoria reservada con *GlobalAlloc*. Esto se utiliza siempre que se quiera acceder al contenido, y asegurar así la disposición del bloque. Esta función retorna el puntero al bloque de memoria indicado por el *handle*.

BOOL GlobalUnlock(HGLOBAL hMem);

Esta función permite liberar la memoria bloqueada con *GlobalLock*. De este modo, el puntero de acceso al bloque de memoria deja de tener sentido, por lo que no debe utilizarse.

HGLOBAL GlobalFree(HGLOBAL hMem);

Esta función libera el bloque de memoria reservado con *GlobalAlloc*, por lo que ya no debe utilizarse el *handle* a dicho bloque.

INICIALIZACIÓN DE LOS DATOS

Una vez que tenemos el *handle* a la memoria donde están los datos del DIB, hay que inicializar algunos campos para mostrar la imagen. En primer lugar se almacena en la variable *m_sizeDoc* el tamaño de la imagen. Para ello se llaman a las funciones *DIBWitdh* y *DIBHeight* del fichero *dibapi.cpp*, que se encargan de obtener a partir del puntero a los datos el ancho y el alto de la imagen respectivamente.

Después de esto, se crea un objeto del tipo *CPalette* para almacenar los colores de la imagen, acción que se

realiza dentro de la función *CreateDIBPalette* del fichero *dibapi.cpp*. Dentro de esta función se fija de nuevo el *handle* a los datos y se inicializa la variable *lpbmi* para acceder a los datos del tipo *BITMAPINFO*.

Primero se obtiene el número de colores llamando a la función *DIBNumColors*, función que lee el campo *biClrUsed*, el cual es cero si el número de colores usados es menor del número de colores por pixel. En caso contrario, se devuelve el número de colores por pixel a partir del campo *biBitCount*.

Después de conocer el número de colores a usar, debemos reservar memoria para almacenar los datos de la paleta. El espacio necesario es el tamaño de la estructura *LOGPALETTE* más el tamaño de la estructura *PALETTEENTRY* por el número de colores. El siguiente paso es bloquear la memoria para almacenar los datos necesarios. Se almacena la versión de la paleta en *palVersion*, el número de colores en *palNumEntries* y, por supuesto, se deben almacenar cada uno de los colores del DIB en la paleta. Para ello se realiza un bucle de 0 a el número de colores, copiando los valores de cada valor RGB sobre las entradas de la paleta y poniendo el valor *peFlags* a cero.

Una vez inicializados los datos de la paleta se debe asignar ésta al objeto *CPalette* mediante la función miembro *CreatePalette*. Una vez asignado, ya se puede liberar la memoria reservada. A partir de este momento, ya están todos los datos listos para mostrar la imagen.

MOSTRANDO LA IMAGEN

Antes de mostrar la imagen debemos de cargar la paleta de colores adecuada, y para ello en la clase que gestiona la ventana principal se atienden los mensajes *OnPaletteChanged* y *OnQueryNewPalette*.

El mensaje *OnPaletteChanged* es enviado por el entorno cuando la ventana con el foco llama a la función *RealizePalette*, lo que implica que la paleta lógica pase a ser la paleta del sistema. Este mensaje permite a las ventanas sin el foco actualizar paletas. Este mensaje es enviado a todas las ventanas de mayor nivel y las ventanas hijas, incluida la que cambió la paleta, por lo que una ventana no debe llamar a *RealizePalette* si detecta que el *handle* de la ventana que cambió la paleta es el suyo propio. En caso de no hacerse así, se entraría en un bucle infinito.

El mensaje *OnQueryNewPalette* es enviado por el entorno cuando una ventana toma el foco, permitiendo a la misma que establezca su paleta como paleta del sistema.

El tratamiento en ambos mensajes es el mismo, se busca la ventana MDI activa, se obtiene el *handle* de la vista asociada con dicha ventana MDI y se le manda el mensaje definido por el usuario *WM_DOREALIZE*.

En el fichero *dibview.cpp* se trata a este mensaje en la función miembro *OnDoRealize*. En esta función se obtiene el *handle* a la paleta cargada en el documento. Para ello se llama a la función miembro *GetDocPalette()* de la clase *CDibDoc*, creada para retornar la dirección de la variable miembro *m_palDIB*, donde se almacena la paleta cargada desde el fichero. Una vez que tenemos el puntero a la paleta, la seleccionamos dentro del contexto de dispositivo de la vista, obtenido desde la clase *CClientDC*. Para seleccionarlo se llama a la función miembro *SelectPalette* pasándole la *palette* y un *boolean* que indica si la paleta es de fondo o no, como función de si la ventana a la que le llega el mensaje es la activa o no. Una vez seleccionada la paleta se llama a la función *RealizePalette*. Esta función devuelve el número de colores que cambian respecto a la paleta anterior, por lo que si este valor es mayor de cero se deben actualizar las vistas llamando a la función *UpdateAllViews()*.

Cuando una ventana necesita actualizar su contenido gráfico, el entorno llama a la función *OnDraw*, función donde vamos a mostrar la imagen. Lo primero que se hace es obtener el *handle* a la zona de memoria donde está la imagen, y para ello se debe obtener primero el puntero al objeto que gestiona el documento, y una vez que tengamos el puntero llamamos a su función miembro *GetHDIB()*. Después de esto, se obtiene el tamaño de la imagen como se ha visto anteriormente. El siguiente paso es llamar a la función *PaintDIB*, función encargada de mostrar el DIB. Los parámetros que se le pasan son el contexto de dispositivo, el rectángulo destino de la imagen, la zona de memoria de la imagen, el rectángulo de la imagen y el puntero al objeto *CPalette* del documento.

Dentro de la función *PaintDIB*, lo primero que se vuelve a hacer es fijar el bloque de memoria. Posteriormente, se obtiene la información de la estructura *BITMAPINFOHEADER* llamando a la función *FindDIBits*. Esta función calcula la dirección de los bits que definen el DIB. Para ello coge el contenido del primer valor *DWORD* de la estructura *BITMAPINFOHEADER*, la cual indica el tamaño reservado por dicha estructura y le suma el tamaño ocupado por las estructuras que definen las entradas de colores, tamaño calculado por la función *PaletteSize*. Esta función calcula el tamaño multiplicando el

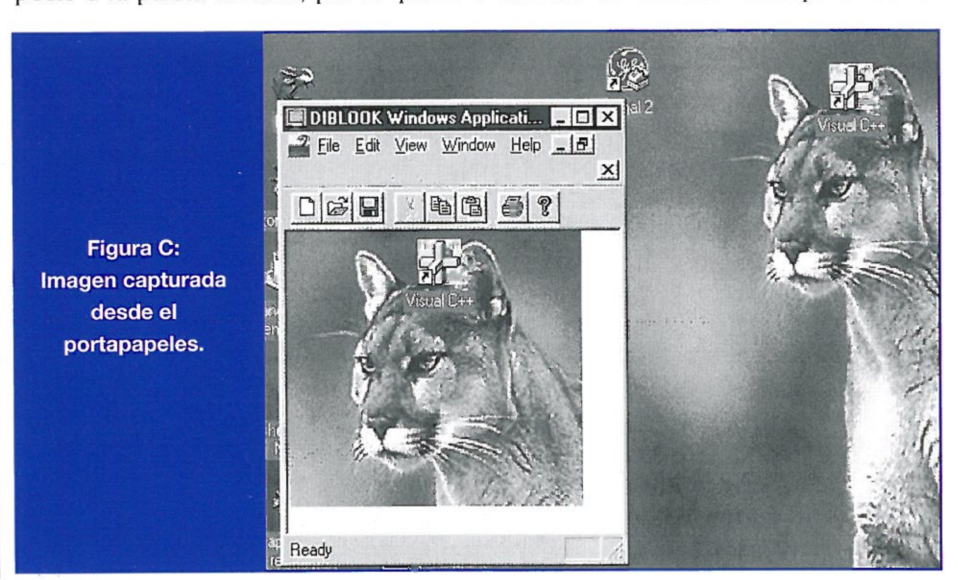


Figura C: Imagen capturada desde el portapapeles.

número de entradas utilizadas por el tamaño de la estructura *RGBQUAD*.

Después de esto, obtenemos el *handle* del objeto *CPalette*, que contiene la paleta de colores del documento, y lo seleccionamos dentro del contexto de dispositivo de la vista con la función *SelectPalette*.

Por último, se muestra la imagen dentro de la vista utilizando las funciones *SetDIBitsToDevice* o *StretchDIBits*, aunque con anterioridad se llama a la función *SetStretchBltMode* para usar el mejor método para ajustar la imagen al contexto de dispositivo.

LA FUNCIÓN SETDIBITSTODE VICE

La función *SetDIBitsToDevice* copia un rectángulo de pixels de un DIB sobre el rectángulo de la ventana indicado. El formato de la función es el siguiente:

*int SetDIBitsToDevice(HDC hdc, int XDest, int YDest, DWORD dwWidth, DWORD dwHeight, int XSrc, int YSrc, UINT uStartScan, UINT cScanLines, CONST VOID * lpvBits, CONST BITMAPINFO * lpbmi, UINT fuColorUse);*

donde:

hdc es el *handle* al contexto de dispositivo.

XDest es la coordenada X donde se muestra la imagen.

YDest es la coordenada Y donde se muestra la imagen.

dwWitdh es el ancho del rectángulo de la imagen original.

dwHeight es el alto del rectángulo de la imagen original.

XSrc es la coordenada X donde comienza la imagen.

YSrc es la coordenada Y donde comienza la imagen.

uStartScan es la línea donde se empieza a escanear la imagen.

cScanLines es la cantidad de líneas a mostrar.

lpvBits es la dirección donde se encuentran los bits de la imagen.

lpbmi es la dirección donde se encuentra la estructura *BITMAPINFOHEADER*.

fuColorUse indica si el campo *bmiColors* de la estructura *BITMAPINFO* contiene entradas RGB o bien índices de una paleta.

LA FUNCIÓN STRETCHDIBITS

La función *StretchDIBits* copia un rectángulo de la imagen sobre el rectángulo seleccionado, ajustando el tamaño de la imagen. Si el rectángulo de destino es mayor que el original, la imagen es expandida, pero si por el contrario es más pequeño, la imagen es comprimida, pero siempre se muestra la imagen completa. El formato de la función es:

*int StretchDIBits(HDC hdc, int XDest, int YDest, int nDestWidth,
int nDestHeight, int XSrc, int YSrc, int nSrcWidth,
int nSrcHeight, CONST VOID * lpBits,
CONST BITMAPINFO * lpBitsInfo, UINT iUsage, DWORD dwRop);*

donde:

hdc es el contexto de dispositivo.

XDest es la coordenada X donde se muestra la imagen.

YDest es la coordenada Y donde se muestra la imagen.

nDestWitdh es el ancho del rectángulo de la imagen original.

nDestHeight es el alto del rectángulo de la imagen original.

XSrc es la coordenada X donde comienza la imagen.

YSrc es la coordenada Y donde comienza la imagen.

nSrcWitdh es el ancho del rectángulo de la imagen original.

nSrcHeight es el alto del rectángulo de la imagen original.

lpBits es la dirección donde se encuentran los bits de la imagen.

lpBitsInfo es la dirección donde se encuentra la estructura *BITMAPINFOHEADER*.

iUsage indica si el campo *bmiColors* de la estructura *BITMAPINFO* contiene entradas RGB o bien índices de una paleta.

DwRop indica cómo los pixels origen van a ser combinados con los pixels en el destino.

Esta función puede crear imágenes espejo respecto al eje X si los signos de *nDestWidth* y *nSrcWidth* son diferentes, y respecto al eje Y si los signos de *nDestHeight* y *nSrcHeight* son diferentes también.

GUARDAR UN DIB EN UN FICHERO

Cuando se pulsa sobre la opción *Guardar* o *Guardar como...* del menú *Fichero*, el entorno llama automáticamente a la función *OnSaveDocument*, función aprovechada para almacenar el DIB en ese momento en memoria sobre un soporte permanente.

En el inicio de la función se abre el fichero indicado por el nombre pasado como parámetro y se llama a la función *SaveDIB* pasándole el *handle* de la memoria que apunta al DIB y una referencia al objeto que gestiona la comunicación con el fichero.

Dentro de la función *SaveDIB* incluida en el fichero *myfile.cpp* se llenan las estructuras *BITMAPFILEHEADER* y *BITMAPINFOHEADER* con los datos adecuados. Lo primero es bloquear el *handle* de memoria del DIB, lo que devolverá la dirección de la estructura *BITMAPINFOHEADER* del DIB. Posteriormente hay que llenar todos los campos de la estructura *BITMAPFILEHEADER*. La primera asignación es el tipo de fichero, lo que implica almacenar en el campo *bfType* la cadena "BM".

La parte más complicada en el proceso de grabación es el cálculo del tamaño del fichero, ya que debe ser muy precisa para que otras aplicaciones que abran el DIB no fallen al cargarlo. Lo primero que hay que hacer es calcular el tamaño ocu-

pado por las estructuras *BITMAPINFOHEADER* y el *array* de estructuras *RGBQUAD*. Este calculo se realiza sumando el primer campo de la estructura *BITMAPINFOHEADER* que indica el tamaño que ocupa la propia estructura más el tamaño de la estructura *RGBQUAD* por el número de entradas de color utilizadas. El calculo del segundo sumando está encapsulado en la función *PaletteSize*, comentada ya anteriormente. El siguiente cálculo pasa por saber el tamaño de los bits que definen la imagen. En este cálculo existen dos partes bien diferenciadas. La primera es si el DIB está comprimido, lo que implica que no se puede calcular y haya que fiarse del contenido del campo *biSizeImage* de la estructura *BITMAPINFOHEADER*. En el segundo caso, sin comprimir, se calcula el tamaño con la ayuda de los campos *biWidth*, *biHeight* y *biBitCount*. El campo *biWidth* indica el ancho del DIB en pixels alineado en *DWORD*, el campo *biBitCount* define el número de bits por pixel y *biHeight* la altura de la imagen en pixels, por lo que el tamaño en bytes será:

```
dwSizeDIBits = ALINEAR_A_DWORD(biWidth * biBitCount) * biHeight
```

donde *ALINEAR_A_DWORD* indica los bytes necesarios para almacenar todos los bits alineados en valores *DWORD*. La operación para realizar este calculo es:

```
res = ((bits) + 31 ) * 32 / 4
```

donde *bits* es el resultado de multiplicar *biWidth* * *biBitCount*, que da el ancho en bits. Al sumarle 31 y dividir por 32 se obtiene el número mínimo de valores *DWORD* necesarios para empaquetar todos esos bits, y por último al multiplicar por 4 se obtiene el número de bytes necesarios. Cabe destacar que la división es entera, por lo que se pierde el resto y no existen decimales.

Una vez hechos estos dos cálculos sólo falta sumar el tamaño de la estructura *BITMAPFILEHEADER*. El resultado obtenido se almacena en el campo *bmfSize*. Los campos *bmfReserved1* y *bmfReserved2* deben asignarse a 0. La última asignación debe ser el *offset* existente desde el origen del fichero hasta los bits que definen el DIB. Este cálculo se hace sumando el tamaño de la estructura *BITMAPFILEHEADER*, la estructura *BITMAPINFOHEADER* y el *array* de estructuras *RGBQUAD*, cálculo ya realizado en las líneas anteriores.

BITMAPFILEHEADER
BITMAPINFOHEADER
RGBQUAD
BITS DE LA IMAGEN

Figura B. Formato de los ficheros DIB.

Después de estas operaciones, se debe guardar en fichero la estructura *BITMAPFILEHEADER*, seguida de la zona de memoria que contiene el DIB con el tamaño ya calculado. Con esto finaliza la grabación de los DIBs en disco.

DIBS Y EL PORTAPAPELES

Otra forma para transferir DIBs es mediante el portapapeles, como ya sabemos, una muy potente herramienta para transferir de forma muy sencilla información entre aplicaciones. Para dar soporte a los DIBs desde el portapapeles, hay que atender los mensajes *OnEditCopy* y *OnEditPaste*.

La función *OnEditCopy* permite insertar el DIB del documento dentro del portapapeles, y para ello hay que realizar los siguientes pasos: Lo primero es abrir el portapapeles con la función *OpenClipboard*. Una vez abierto, se borra su contenido y se llama a la función *SetClipboardData*, pasándole como primer parámetro el tipo de objeto a insertar, en este caso es *CF_DIB* al tratarse de un DIB, y como segundo parámetro una copia del *handle* a la memoria del DIB. Para realizar la copia se utiliza la función *CopyHandle* contenida en el fichero dibapi.cpp. Esta función reserva un bloque de memoria global del mismo tamaño que el dado, fija ambos y realiza una copia del contenido. Luego los desbloquea y devuelve el *handle* del bloque de memoria copiado. Después de todo esto, sólo resta cerrar el portapapeles para finalizar la operación. En la función *OnEditPaste* se realiza la operación complementaria, es decir, obtener el contenido del portapapeles y almacenarlo en la memoria de la aplicación. Las operaciones a seguir pasan también por abrir el portapapeles, copiar el *handle* al DIB obteniendo anteriormente el *handle* con la función *GetClipboardData* pasándole como parámetro el tipo de objeto a obtener, en este caso CF_DIB. Una vez comprobado que el *handle* es válido, se obtiene el objeto *CDibDoc* asociado a la vista para poder acceder a las funciones *ReplaceDIB* y *InitDIBData*. La función *ReplaceDIB* se encarga de liberar la memoria asociada con la variable miembro *m_hDIB* y asignarla al nuevo *handle*. Después de esto, se notifica al documento el cambio del contenido con *SetModifiedFlag*, se actualizan las barras de *scroll*, se manda el mensaje para actualizar la paleta de colores y se actualizan todas las vistas de la aplicación.

CONCLUSIÓN

En este artículo se pretende ilustrar el modo de trabajo a seguir con los DIBs, desde la carga de un fichero, el trabajo con la paleta de colores, la forma de mostrar la imagen y la forma de guardarla en disco. Para ello se ha descrito uno de los programas ejemplo existentes dentro de Visual C++ 4.0, el cual muestra de forma inmejorable todos estos pasos, que han sido descritos a lo largo de este artículo.

ARCHIVOS (II)

José C. Remiro

En el número anterior se establecieron las bases para la implementación de un método de acceso a la información albergada en un archivo directo a través de un archivo de índices. Se discutió su estructura y se procedió a definir las operaciones de creación e inserción. En la presente entrega se completarán las operaciones más usuales a realizar y se presentará otro método para acceder a la información de forma similar al método de archivos índice, pero sin utilizar este archivo auxiliar.

Este artículo trata de completar las operaciones sobre los archivos indexados y presentar otro tipo de organización para un archivo, de tal forma que el acceso a los registros sea de tipo directo a través de una clave: el método Hash.

OTRAS OPERACIONES

La operación de eliminar un registro del archivo principal debe ser diseñada cuidadosamente, pues no está permitido en un archivo de acceso directo eliminar un registro. Para solucionar este problema sobre el archivo de índices se procede a realizar un proceso de "borrado lógico", de tal forma que se indique que la información de un determinado registro no se encuentra disponible cuando se intente realizar una operación sobre él.

La solución más usual consiste en añadir un campo al registro del archivo de índices, de tal forma que se distinga a través de él si el registro correspondiente se encuentra disponible o no. Así, en los programas que acompañan al artículo se ha añadido al registro del archivo de índices un campo denominado *borrado*, de tipo *boolean*, de tal forma que si contiene el valor *true* indicará que el registro del archivo principal al que apunta no se encuentra disponible (a pesar de contener información). Si, por el contrario, el campo contiene el valor *false*, entonces el registro sí está disponible. Este campo no se podría haber situado en el archivo principal, pues el método de búsqueda utilizado no permite disponer de los registros dentro del archivo de índices con igual clave. Además, el método sería más ineficiente, pues tras haber realizado el proceso de búsqueda sobre el archivo de índices se debería haber accedido al archivo principal para comprobar finalmente si el registro se encontraba disponible.

Así, cuando un usuario decide borrar un registro se debe comprobar en primer lugar si el registro existe, y de ser así asignar el valor *true* al campo *borrado*.

La anterior operación, que parece tan sencilla, tiene una complicación adicional: supóngase que se borra lógicamente un registro y posteriormente el usuario desea introducir uno nuevo con la misma clave principal del registro anteriormente borrado. Si se procede a realizar una inserción de la forma habitual, en el registro de índices se encontrarán dos registros con la misma clave y, por tanto, cuando más tarde el usuario desee recuperar la información, si el proceso de búsqueda encuentra antes el registro del archivo de índices que indica que dicha información está marcada para borrar, al usuario se le transmitirá que la información no está disponible, aún estando almacenada en un registro.

Por lo expuesto anteriormente, es obligatorio que el proceso de altas de registros busque en primer lugar si en el archivo de índices ya hay un registro con dicha clave. Si es así y no está borrado, entonces el proceso de altas no se podrá realizar, pues el archivo de índices es un índice principal y las claves deben ser únicas. Pero si está borrado, el proceso de altas sí se podrá llevar a cabo, pero con una diferencia: la información a almacenar en el archivo principal debe situarse en el registro

CUADRO 1

```
program bajas;
 uses
 listas, indice, auxiliar;
 var
  reg_m_aux: reg_termino;
  resp_sn: boolean;
 begin
  pantalla_titulos (titulo_bajas);
  resp_sn:= pregunta(columna, fila, preg_otra_baja);
  while resp_sn
     do
     begin
      obten_clave (reg_m_aux);
      if clave_encontrada(arch_ind, l_temporal, reg_m_aux.palabra)
        then
         begin
          obten_registro_principal (reg_m_aux);
          visualiza_resto_informacion (reg_m_aux);
          resp_sn:= pregunta(columna, fila, preg_borrar);
          if resp_sn
            then
              marca_registro_borrado
         end
        else
         visualiza_y_para (columna, fila, no_existe_pulsar);
      pantalla_titulos (titulo_bajas);
      resp_sn:= pregunta(columna, fila, preg_otra_baja);
     end;
 end.
```

al que apunta el registro del archivo de índices y modificando el valor del campo *borrado* a *false*.

Como consecuencia del diseño de la operación de borrado, a lo largo de la vida de los archivos ocurrirá en ocasiones que el archivo de índices contenga gran cantidad de registros con el campo *borrado* a *true*, con el consiguiente desaprovechamiento del espacio de almacenamiento. Por tanto, habrá que diseñar un procedimiento para eliminar físicamente aquellos registros que estén marcados como borrados, pero conservando la relación que une al archivo de almacenamiento principal y al archivo de índices.

En el cuadro 1 se puede ver el programa BAJAS, que marca un registro indicando que su información no está disponible. Se trata de un proceso iterativo que puede detener el usuario cada vez que se le pregunta si desea borrar algún otro registro. Cada vez que se introduce una clave principal a buscar, el proceso llama a la función *clave_encontrada*, que se encarga de buscar en la lista temporal y en el archivo de índices si hay algún registro con dicha clave. Si este es el caso, dicho proceso habrá almacenado el número de registro en la variable *pos_aux* de la unidad *indice* si la clave ha sido encontrada en el archivo de índices, o bien en la variable *ptr_pos_aux* si la clave ha sido encontrada en la lista temporal de registros.

Posteriormente, se procede a la visualización de la totalidad del registro y a pedir la conformidad del usuario para marcar el registro como borrado. Si la respuesta es afirmativa, se procede a la llamada del procedimiento *marca_registro_borrado*, que accede bien al registro del archivo índice que contiene la clave a borrar, bien al nodo de la lista temporal gracias a la información almacenada en las variables mencionadas anteriormente.

Hay que hacer notar que la lista temporal, mientras se ejecute el programa *bajas*, estará siempre vacía, pues es necesario dar de alta a un registro para que dicha lista contenga algún nodo. Sin embargo, el programa comprueba si la clave se encuentra en dicha lista. Esto es así para permitir que el proceso

CUADRO 2

```
procedure elimina_marcados;
 var
  reg_aux_ind: reg_indice;
  reg_aux_prn: reg_termino;
  f_aux_ind: arch_indice;
  f_aux_prn: arch_principal;
  pos: longint;
 begin
  assign(f_aux_ind, 'tempind.tmp');
  rewrite(f_aux_ind);
  assign (f_aux_prn, 'tempprn.tmp');
  rewrite(f_aux_prn);
  pos:= 0;
  while not eof(arch_ind)
    do
     begin
      read(arch_ind, reg_aux_ind);
      if not reg_aux_ind.borrado
        then
         begin
          seek(arch_prn, reg_aux_ind.posicion);
          read(arch_prn, reg_aux_prn);
          write(f_aux_prn, reg_aux_prn);
          reg_aux_ind.posicion:= pos;
          write(f_aux_ind, reg_aux_ind);
          pos:= pos + 1
         end
     end;
  close (arch_prn);
  close (arch_ind);
  close (f_aux_prn);
  close (f_aux_ind);
  erase (arch_prn);
  erase (arch_ind);
  rename (f_aux_prn, tray_arch_prn);
  rename (f_aux_ind, tray_arch_ind)
 end;
```

CUADRO 3

Algoritmo para eliminar los registros del archivo principal que no se encuentran en el archivo de índices.

```
inicio
 abrir (arch_principal);
 abrir (arch_indice);
 abrir (arch_aux);
 mientras no fin_archivo(arch_principal)
   hacer
     leer (arch_principal, reg_principal);
     obten_clave (reg_principal);
     Si existe (clave, arch_indice)
       entonces
         escribir (arch_aux, reg_principal);
 fin-mientras
 cerrar (arch_principal);
 cerrar (arch_indice);
 cerrar (arch_aux);
 renombrar (arch_aux);
fin.
```

representado por el programa BAJAS sea lo más genérico posible y pueda fusionarse con el resto de programas que lo acompañan para poder formar un programa mayor dedicado al mantenimiento integral del archivo, por ejemplo uniéndolos a través de un menú.

Para la eliminación definitiva de los registros marcados como borrados se ha realizado el procedimiento *elimina_marcados* (ver cuadro 2). Éste crea dos archivos auxiliares: uno que representará al nuevo archivo de índices y otro al principal. Cuando finaliza el procedimiento, elimina los anteriores y se procede a renombrarlos con el nombre de los archivos originales. Este procedimiento realiza su tarea de forma muy sencilla: explora secuencialmente el archivo de índices; si el registro que está tratando actualmente está marcado como borrado, entonces lo ignora; por el contrario, si no está marcado lee, gracias a su campo *posicion*, el registro del archivo principal correspondiente, grabando éste en el nuevo archivo principal y guardando además en el nuevo archivo de índices el registro de índices, pero modificando la posición que indica dónde se encuentra el registro del archivo principal al que apunta. Para conocer esta posición se utiliza un contador (inicialmente puesto a 0) que se va incrementando a medida que se almacenan los registros no borrados. Al final de este proceso los dos archivos se encuentran libres de los registros marcados para borrar y, además, ambos se encuentran ordenados por la clave.

DEGENERACIÓN DE ÍNDICES

Uno de los principales problemas que presenta la indexación de un archivo es la posible degeneración del archivo de índices. Esto ocurre cuando la información almacenada en el archivo principal no se corresponde con la información que se puede encontrar en el archivo de índices. Supóngase la siguiente situación: se ha procedido a dar de alta varios registros en el archivo principal, puesto que se ha decidido almacenar al final de dicho archivo todos los registros que se han dado de alta y, en una lista temporal, todos los registros del archivo de índices que indican dónde se encuentran los nuevos registros. Un fallo de alimentación de la corriente eléctrica mientras se procede a realizar este proceso provocará la pérdida de información relativa a los registros dados de alta, pues éstos no aparecerán reflejados en el archivo de índices.

Como consecuencia de la anterior situación, cuando el usuario intente buscar un registro que ha dado de alta no podrá encontrarlo, pues su clave no está reflejada en el archivo de índices. Para solucionar este problema habrá que incluir aquellas claves de registros que, encontrándose en el archivo principal, no se encuentran en el archivo de índices.

El algoritmo que resuelve este problema se encuentra reflejado en el cuadro 3. Se procederá a leer secuencialmente el archivo de almacenamiento principal. Para cada registro se obtendrá su clave y se buscará en el archivo de índices (recuérdese que esta búsqueda se realiza sobre un archivo ordenado por dicha clave). Si se encuentra un registro con dicha clave, entonces es correcta la información almacenada, pero si no se encuentra se debe proceder, o bien a eliminar el registro del archivo principal, o bien a la inserción de un nuevo registro en el archivo de índices, o bien a la eliminación del registro del archivo principal, guardando dicha información en un archivo auxiliar para que posteriormente el usuario pueda decidir qué registros deben darse de alta.

Como puede verse en el cuadro 3, el proceso anteriormente descrito es muy similar al proceso de eliminación de los registros marcados para borrar, pero el tratamiento secuencial está guiado esta vez por el archivo principal.

HASHING

La importancia de los archivos indexados reside en que es un método capaz de localizar un registro a través de una clave. Sin embargo, como se ha puesto de manifiesto anteriormente, su gestión presenta una serie de inconvenientes, siendo los más importantes la degeneración de sus índices y el hecho de que para su implementación haya que recurrir a la gestión conjunta de archivos.

Existe otro método para localizar un registro conocida su clave sin tener que

CUADRO 4

clave	Caso
transformación con códigos ASCII	C → ord('C') - ord('A') = 67 - 65 = 2 a → ord('A') - ord('A') = 65 - 65 = 0 s → ord('S') - ord('A') = 83 - 65 = 18 o → ord('O') - ord('A') = 79 - 65 = 14
Agrupación de dos en dos	2 + 0 = 2 posición 0 ← 18 + 14 = 32 posición 1 ←
Producto por los pesos según posición	$2 \times 2^0 + 32 \times 2^1 = 2 + 64 = 66$
División por el tamaño del archivo	66 mod 71 = 66 (posición que ocupará el registro con clave "Caso").

Cálculo de la posición de un registro conocida su clave

recurrir a la creación de un archivo auxiliar de índices: el método *hashing*. Este método se basa en la construcción de una función que transforme la clave de un registro en un número entero que se encuentre entre cero y el número total de registros que puede almacenar un archivo previamente creado, de tal forma que para localizar un registro conocida su clave baste con aplicar la función a dicha clave, obteniéndose de esta forma la posición que ocupa. Aunque el método parece sencillo según lo anteriormente expuesto, también presenta algunos problemas.

Previamente a la utilización del método *hashing* se debe establecer una estimación del número de registros de que va a disponer el archivo, pues éste se creará manteniendo un número fijo de registros y a medida que el usuario lo solicite se reescribirán éstos. El segundo paso a realizar es la construcción de una función que, dada una clave, se obtenga un número de registro para la inserción o recuperación del correspondiente registro y, por último, cómo se realizará la gestión de la colisión de claves, entendiendo por ésta la correspondencia de un mismo número de registro a claves diferentes.

CONSTRUCCIÓN DE LA FUNCIÓN

La función más sencilla que puede construirse para localizar un registro es la función identidad, es decir, asociar como clave de un registro la posición que éste ocupa dentro del archivo. En verdad, esto no soluciona el problema, pues en la mayoría de las situaciones la clave no es ni siquiera de tipo numérico, sino alfanumérico, y por tanto habrá que pensar primero en trasladar esta información alfanumérica a numérica. Supóngase que se desea aplicar el método *hashing* al mismo tipo de información que en el ejemplo presentado para la construcción del archivo de índices, es decir, la clave de un registro es una cadena de caracteres de longitud 15. Una posible transformación a clave numérica consistiría en tomar el código ASCII de cada uno de los caracteres que forma la cadena y realizar operaciones aritméticas sobre ellos. En particular, sería interesante tener en cuenta la posición que ocupa cada uno de ellos dentro de la cadena, de tal forma que claves que dispongan de los mismos caracteres, pero en diferente orden, no tengan asociado el mismo número de registro para así evitar colisiones.

Para resolver el anterior problema es preciso asociar una serie de pesos dependiendo de la posición que ocupen los caracteres de la clave. También hay que tener especial cuidado en que el número así obtenido no supere la capacidad de almacenamiento de una variable de tipo *longint* donde se almacenará dicho resultado. Por último, y en lo que se refiere a la construcción de la función, habrá que corregir el número obtenido para que éste se encuentre entre cero y el número total de registros establecido inicialmente. Para realizar este proceso se puede dividir por el número total de registros el número que se obtuvo al final del anterior proceso y tomar el resto. Con esto se asegura que el número obtenido finalmente se encuentre entre cero y el tamaño (en registros) del archivo.

En el cuadro 4 se puede observar el algoritmo seleccionado como ejemplo, junto con un ejemplo para la transformación de la clave. El proceso a seguir es el siguiente:

- Se transforman todos los caracteres (se supondrán alfabéticos) a caracteres en mayúscula.
- Para cada uno de ellos se calcula la diferencia de su código ASCII con el código ASCII de la letra A (la de menor código).
- Se suman de dos en dos los valores anteriormente calculados y se les multiplica por una potencia de dos, dependiendo del lugar que ocupen.
- Se procede a realizar la suma de los valores anteriormente obtenidos.
- Se procede a dividir el anterior resultado por 97 (se supondrá que el archivo es de 100 registros), el resto será la posición que ocupe el registro que disponga de la clave inicial.

Para seleccionar el tamaño del archivo se debe estimar calculando aproximadamente el número de registros que albergará y aumentando éste número en un 30% para poder gestionar de forma adecuada las colisiones que, seguramente, se producirán.

OPERACIONES

Para realizar la gestión de los registros hay que incluir dentro de éstos un campo para distinguir varias situaciones que pueden darse tras realizar las operaciones asociadas al archivo. Supóngase que se desea introducir un nuevo registro. La primera acción a realizar consiste en aplicar la función de transformación a la clave para obtener el número de registro donde se debería grabar la información. Una vez calculada se procede a leer el registro de la posición calculada. Éste puede tener estado "libre", es decir, no ha sido ocupado con antelación. Si este es el caso, se procede a introducir la información en dicho registro y a establecer su estado como "ocupado", indicando que

CUADRO 5

```
procedure busca_clave_duplicada (pos: longint;
         clv: clave;   var existe: boolean);
 var
  aux_pos: longint;
  reg_aux: reg_hash;

  procedure auxiliar_busqueda;
   begin
    seek(arch_hash ,aux_pos);
    read(arch_hash, reg_aux);
    while (not existe) and (reg_aux.estado <>
                              libre)
       and (aux_pos < long_arch)
      do
      if (reg_aux.estado = ocupado) and
        (reg_aux.termino = clv)
       then
        existe:= true
       else
         begin
          aux_pos:= aux_pos + 1;
          seek(arch_hash, aux_pos);
          read(arch_hash, reg_aux)
         end;
      end; {del procedimiento auxiliar_busqueda}

 begin
   existe:= false;
   aux_pos:= pos;
   auxiliar_busqueda;
   if (not existe) and (reg_aux.estado <> libre)
    then
     begin
      aux_pos:= 0;
      auxiliar_busqueda;
     end;
   end;
```

contiene información significativa. Pero si se ha producido una colisión de claves, es decir, dicho registro tenía el estado "ocupado", habrá que buscar algún otro registro con estado "libre" para poder almacenar la información. En efecto, se procede a la inspección de los registros que se encuentran inmediatamente a continuación hasta encontrar uno que se encuentre en estado "libre". Una vez encontrado se procederá a la inserción de la información y a establecer su campo de estado a "ocupado". En este caso habrá que tener especial cuidado cuando se llegue al último registro. Si esto ocurre, se procederá a continuar la inspección en el primer registro y, si fuera, necesario los siguientes.

Para eliminar un registro, se comenzará de nuevo aplicando la función de transformación a la clave para posteriormente acceder al registro cuyo número ha sido calculado. Se procederá a leer dicho registro y, si ambas claves coinciden, entonces se procede a su eliminación lógica, estableciendo "borrado" como estado del registro. Si las claves no coincidiesen, entonces habría que consultar en orden secuencial todos los registros, comprobando cada una de sus claves hasta encontrar uno que disponga de igual clave (si esto ocurre, entonces se habrá encontrado) o hasta que se encuentre un registro con estado "libre", en cuyo caso el registro no figura en el archivo (por tanto, no podrá ser borrado, pues no existe).

Es posible preguntarse por qué el estado de los registros que se dan de baja debe ser "borrado" en vez de "libre". La respuesta es sencilla y se comentará mediante un ejemplo: supóngase que se da de alta un registro, pero de tal forma que ya hay otro ocupando su lugar en el archivo (es decir, se ha producido una colisión). Entonces se procederá a buscar el siguiente registro con estado "libre" para albergarlo. Supóngase que ahora se elimina el registro que estaba almacenado en la posición que debería haber ocupado el último registro insertado. Si a éste se le asigna el estado "libre", implicaría que el registro que se insertó en último lugar no está en el archivo, pues de estar, estaría allí.

Para evitar la anterior situación, el estado "borrado" indica que la búsqueda debe continuar hasta encontrar el registro que se está buscando, pues allí en su momento hubo un registro que desplazó al que se desea buscar, pero éste último fue eliminado y más adelante podría ser encontrado.

Por tanto, los estados asociados a los registros pueden ser:

"Libre", cuando el registro no ha sido utilizado nunca. Este estado será el estado inicial de todos los registros cuando se cree el archivo. Además, este estado servirá para detener el proceso de búsqueda de un registro.

- "Borrado", cuando se elimine un registro. Este estado, además de indicar que se debe continuar el proceso de búsqueda, sirve para indicar que se puede utilizar para realizar un alta sobre él si fuera necesario.
- "Ocupado" indica que el registro contiene información significativa para el usuario, ayudando este estado a la búsqueda de información.

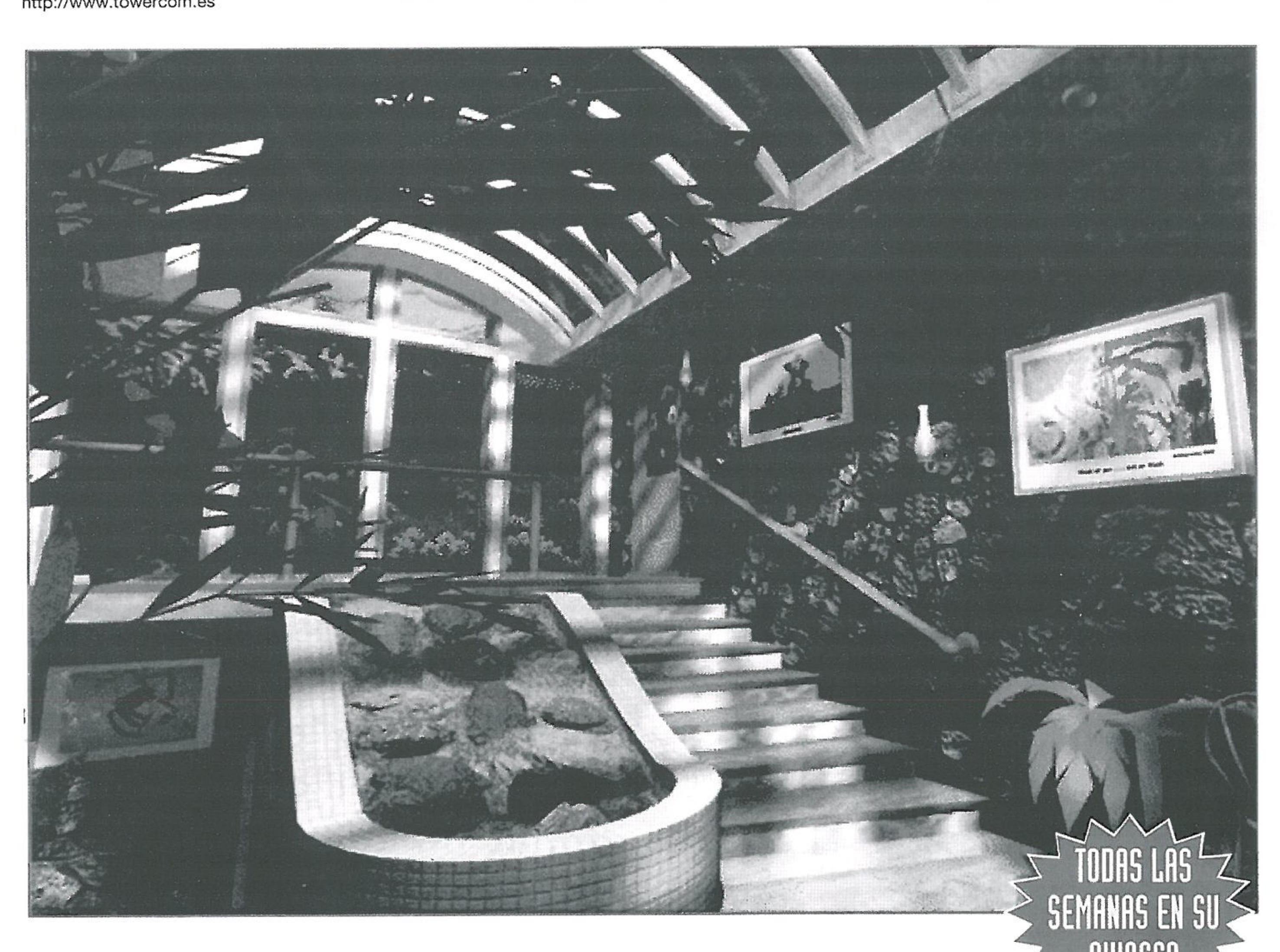
En este punto ya es posible entender la selección de la función de transformación de claves de la forma en que se ha hecho, pues si ocurre una colisión no sólo se desplazan los registros que colisionan, pues éstos se almacenan en registros vecinos, provocando aún más colisiones. Además, la búsqueda que se realiza para localizar un registro cuya clave ha colisionado con otra obliga a la lectura de varios registros para su localización. Por eso se decidió dejar un 30% más de registros que los que se iban a ocupar para que la localización de un registro no involucre, en el peor de los casos, a más de tres o cuatro registros. La selección de un número primo cercano al tamaño del archivo distribuye mejor las claves a lo largo del mismo, de ahí que se seleccionase el número 97 como divisor (el número primo más cercano a 100).

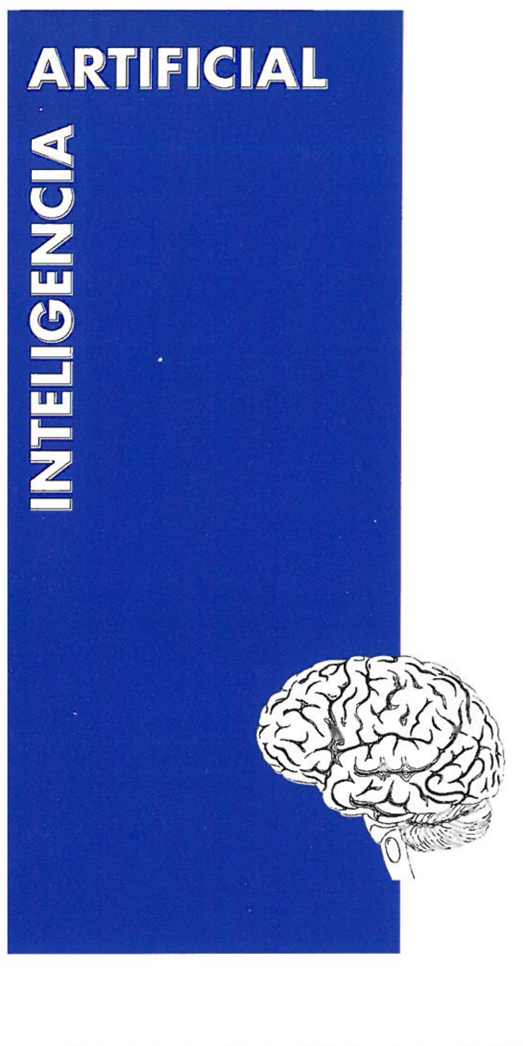
En el cuadro 5 se puede ver el procedimiento *busca_clave_duplicada*, siendo una buena muestra de la filosofía del método *hashing*. Este procedimiento, dada una clave, una posición y una variable booleana, busca el registro con clave dada, a partir de una posición (la que correspondería al registro de estar ésta almacenada en el archivo). Si la encontrase devuelve el valor *true* en la variable booleana, devolviendo *false* en cualquier otro caso. Básicamente, se trata de una búsqueda secuencial hasta que, o bien se encuentra un registro cuyo estado es "libre" (no existe un registro con igual clave), o bien cuando se encuentra un registro con estado "ocupado" con igual clave (se ha encontrado el registro buscado). Es importante darse cuenta de cómo se ignoran los registros con estado "borrado", pero a la vez cómo son utilizados para continuar la búsqueda. Además, el procedimiento ha de tener en cuenta que es posible buscar más allá del último registro. Cuando el procedimiento llega a éste y debe seguir buscando, entonces comienza la búsqueda, pero en el primer registro del archivo. Puesto que interesa decidir rápidamente si un registro se encuentra o no en el archivo, y este proceso se ve retardado por el número de lecturas que hay que realizar, es aconsejable disponer de suficientes registros vacíos para que esto no ocurra. Además, este procedimiento entraría en un bucle infinito si no hay al menos un registro con estado "libre".

COMENTARIOS

En el CD-ROM que acompaña a la revista se incluye una nueva versión de la *unit indice*, en la que se han incluido nuevos procedimientos para la gestión de archivos índice. Además, se incluye la *unit hash*, que contiene diversos procedimientos para la construcción de programas que utilicen dicho método, pudiendo seguir como guía el programa *altahash* para la construcción del resto de las operaciones que se pueden aplicar en el método *hash*.

Se espera que este artículo sirva como referencia básica para la utilización de estos métodos.





INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL

Ramiro Carballo

Con un nombre tan presuntuoso surgió la *Inteligencia Artificial* a la par que la Informática (antes, según las consideraciones), no se sabe si ciencia o ingeniería, y con el ambicioso objetivo de imitar la capacidad intelectual del hombre. Lo cierto es que realmente este término ha sido polémico desde sus orígenes, siempre asociado a la ciencia ficción y al morbo de la rebelión de las máquinas contra sus creadores.

Esa era la visión que teníamos la mayoría de nosotros: unos pensábamos que era una pseudociencia que no llevaría a ningún lugar, ya que no se concibe la inteligencia sin, al menos, algún soporte químico o biológico; otros, los que habíamos hojeado alguna publicación acerca del tema, pensábamos que era el resultado de una telaraña de teoremas tejida por sesudos investigadores de la lógica formal y el pensamiento humano, y que, muy lejos de lo que pueden alcanzar nuestras "entendederas", nos sería de poca utilidad en nuestras actividades relacionadas con la programación.

¿Quién no ha intentado programar alguna vez un juego?. Y una vez conseguido y probadas unas cuantas pantallas, ¿quién no ha deseado dotar al recién nacido de un poco de estrategia para poder no sólo competir con la habilidad y los reflejos del jugador de turno, sino también intentar superar al contrario en inteligencia?.

¿Cuántas veces el lector ha pensado alguna vez en realizar un programa de ajedrez o (los menos ambiciosos), de las tres en raya, y ha desistido al no encontrar la manera de traducir en código su experiencia en el juego?.

¿Quién no ha soñado que su aventura conversacional hiciese justicia a su calificativo y no se tuviese que restringir al limitado conjunto de verbos y nombres de cada escena?.

Se espera, de esta manera, que los próximos artículos de esta serie desvelen algoritmos, herramientas y aplicaciones que hagan cambiar al lector su visión de la I.A. y que permitan ahorrar horas de programación al atacar ciertos problemas desde nuevas perspectivas.

QUÉ NO ES LA I.A.

Debemos empezar aclarando que el hecho de programar en *LISP* o en *PROLOG*, por ejemplo, no implica estar haciendo I.A.

En realidad, hacemos I.A. cuando intentamos programar una máquina para que haga "algo más de lo que le hemos dicho explícitamente que haga". Es decir, frente a una partida de ajedrez, pretenderemos que la máquina cambie su estrategia a medida que cambia el juego, o utilice reglas heurísticas (trucos como que es mejor no mover el rey a no ser que te den jaque), o que tenga capacidad para aprender de la experiencia.

Hacemos I.A. cuando intentamos programar una máquina para que haga "algo más de lo que le hemos dicho explícitamente que haga

No es I.A., por ejemplo, *ELIZA*, un programa diseñado para simular el diálogo entre un psicoanalista y un paciente, en el que el programa analiza la cadena de entrada que le da el usuario (el paciente) buscando unas palabras clave. Si halla alguna de ellas, responde de una

forma predeterminada utilizando parte de la entrada recibida por el paciente, como se muestra en la figura 1.

El programa es divertido pero las respuestas están determinadas a priori (se incluye un ejemplo implementado en Basic en el CD-ROM, tanto para GWBasic o QBasic, y en ambas versiones el lector puede inspeccionar el código y llegar a sus propias conclusiones). En un programa que utilice técnicas de I.A., el propio programador se puede ver sorprendido por los resultados: no es raro que la máquina, con capacidad de aprendizaje, acabe jugando mejor al ajedrez que el propio creador.

Figura 1: "Eliza es un ejemplo de lo que no es Inteligecia Artificial

Eliza: COMO TE LLAMAS?
Paciente: Pepe
Eliza: BIEN, HOLA Pepe

Paciente: HOLA
Eliza: Pepe, BIEN,BIEN...PROSIGUE.

Paciente: PUES NO SE
Eliza: PORQUE ERES TAN NEGATIVO?

Paciente: CREO QUE ESTOY EQUIVOCADO
Eliza: QUE SE SIENTE CREYENDO QUE ESTAS EQUIVOCADO?

Paciente: ESTOY SEGURO
Eliza: CUANTO TIEMPO HAS ESTADO SEGURO?

ÁREAS DE LA I.A.

Podemos decir que la I.A. empezó a dar resultados cuando, en 1956, un programa encontró la demostración de un teorema de lógica matemática, o cuando *GPS (General Problem Solver)* resolvió problemas estableciendo metas y generando el plan para alcanzarlas.

Pero ya en los años 40 se intentaba imitar la inteligencia humana. Por un lado, el médico *Mc Cullogh* y el matemático *Pitts* modelizaron las redes de neuronas artificiales (copias electrónicas de las biológicas), pretendiendo conseguir un simulador de la actividad cerebral.

Comienza así el conexionismo, que podríamos llamar I.A. de bajo nivel (es como hablar de puertas lógicas en Informática). Debemos decir que en redes de neuronas se consiguen grandes soluciones, sobre todo para aquellos problemas que, por su complejidad, resultan "improgramables" (reconocimiento de rostros, por ejemplo).

Frente a esta corriente podríamos considerar el simbolismo, que pretende imitar el comportamiento del pensamiento. Es más sencillo de entender porque se intenta que la máquina solucione los problemas de la manera más parecida posible a como lo haría el ser humano.

Tanto las soluciones conexionistas como las simbolistas dan resultados en distintas ramas de la I.A., como el estudio del lenguaje natural (el sueño de que la máquina nos obedezca en nuestra forma normal de expresarnos y nos responda igualmente), la ingeniería del conocimiento (que a través del aprendizaje constituye sistemas expertos capaces de sustituir a expertos humanos en muchos aspectos), el reconocimiento de formas (desde el reconocimiento de caras humanas hasta el software de OCR que usamos en la oficina) o la planificación de tareas (capaz de darnos la solución a un problema dado, indicándonos el camino a seguir para llegar a ella).

LOS LENGUAJES DE LA I.A.

La I.A. ha estado tradicionalmente "programada" en dos lenguajes: *Lisp* y *Prolog*. Lisp está clasificado como un lenguaje de programación funcional, donde la recursividad es la nota dominante. Prolog intenta programar directamente con las reglas de la lógica, con deducciones o silogismos.

Debe quedar claro que éstos no son los dos únicos lenguajes de propósito específico (aunque no exclusivo) para la I.A.. Además, el lector debe tener presente que los algoritmos y técnicas enunciados en I.A. se

Figura 2: "Ejemplo del código fuente de un programa en Lisp

```
(defun rewrite (l)
  (cond ((atom l) l)
        ((equal (car l) 'not)
         (list 'nand (rewrite (cadr l)) T))
        ((equal (car l) 'and)
         (list 'nand
               (list 'nand
                (rewrite (cadr l))
                (rewrite (caddr l))) T))
        ((equal (car l) 'or)
         (list 'nand
                (list 'nand (rewrite(cadr l)) t)
                (list 'nand (rewrite(caddr l)) t)))
        ((equal (car l) 'xor)
         (list 'nand
               (list 'nand
                     (list 'nand
                     (list 'nand (rewrite (cadr l)) t)
                     (list 'nand (rewrite (caddr l)) t))
               (list 'nand
                     (rewrite (cadr l))
                     (rewrite (caddr l))))
                t))
        (t (list 'error l))))
 (pprint (rewrite '(not (and a (xor a b))))))
```

pueden implementar en cualquier lenguaje: de hecho, será habitual que, en esta serie de artículos, presentemos la codificación en C como alternativa a ejemplos en Lisp o en *Prodigy* (herramienta que será presentada posteriormente).

Como lenguajes ideales para I.A. deberíamos enumerar todos los lenguajes funcionales en si: *scheme, hope, perl, ML (CAML, SML)*, etc... Esto es debido a la facilidad que tiene la programación funcional para representar algoritmos en los términos en los que se lo suele plantear una persona. Ejemplo: si se va a tratar cada elemento de una lista de una determinada manera, no es necesario que se tengan presente en todo momento los elemen-

Será habitual que, en esta serie de artículos, presentemos la codificación en C como alternativa a ejemplos en Lisp o en Prodigy

tos de la lista. El problema se solucionará eficazmente en dos pasos:

a) Tratar el primer elemento.

b) Solucionar el mismo problema con la lista restante (hasta que la lista restante se vacíe).

Lisp, por ejemplo, es un lenguaje simbólico (en lugar de manipular datos numéricos trabaja con representación abstracta), de proceso de listas (*List Processing*), interpretado, compacto y, sobre todo, flexible: tanto los datos como el código tienen la misma estructura, se pueden pasar funciones como parámetros, etc...

Estos lenguajes funcionales son sencillos de utilizar pero, tal vez, algo complicados conceptualmente para un programador acostumbrado a C o Pascal. Por eso se quiere presentar, en esta serie, la herramienta *Prodigy 4.0*, que permitirá implementar la mayoría de las técnicas que veamos en cada entrega sin necesidad de aprender Lisp, Prolog o cualquier otro lenguaje con-

```
                 Copyright (c) Bruno Haible, Michael Stoll 1992, 1993, 1994
> (domain 'bloques)
;; Loading file C:\CLISP\PRODIGY\DOMAINS\BLOQUES\DOMAIN.LIS ...
;; Loading of file C:\CLISP\PRODIGY\DOMAINS\BLOQUES\DOMAIN.LIS is finished.Runni
> (problem 'suss)
;; Loading file C:\CLISP\PRODIGY\DOMAINS\BLOQUES\PROBS\SUSS.LIS ...
;; Loading of file C:\CLISP\PRODIGY\DOMAINS\BLOQUES\PROBS\SUSS.LIS is finished.
> (run)
Solution:
        <quitar c a>
        <dejar c>
        <levantar b>
        <poner b c>
        <levantar a>
        <poner a b>
#<PRODIGY result: T, 1.15 secs, 88 nodes, 1 sol>
```

Figura 3: "Volcado de pantalla de una ejecución de Prodigy 4.0 en un intérprete de Lisp para D.O.S

creto (de poca demanda en el mercado laboral).

Además, el objetivo de esta serie de artículos es que el programador aplique nuevas técnicas para solucionar problemas habituales sin necesidad de cambiar de lenguaje. El uso de lenguajes o herramientas específicas pretende sólo facilitar la compresión de nuevos conceptos.

La programación funcional tiene facilidad para representar algoritmos en los términos en los que se lo suele plantear una persona

Sin embargo, se puede utilizar la comunicación lector-autor para opinar sobre si se consideraría o no oportuno una miniserie sobre Lisp o Prolog. A disposición del lector se ofrece el E-mail del autor. También se puede opinar por carta a la dirección de la revista.

Esta interactividad permitiría dar el curso bajo demanda, por petición de los lectores.

De todos modos, a lo largo de la serie se proporcionarán, en el CD-ROM, intérpretes y compiladores de Lisp y Prolog, la herramienta Prodigy 4.0 con manuales y una versión HTML de un completo manual de *Common Lisp*.

PRODIGY 4.0

Durante parte de esta serie de artículos dedicados a la I.A. nos va a acompañar Prodigy 4.0, herramienta diseñada para investigar en el terreno de la planificación, el aprendizaje automático y la adquisición de conocimiento, que también puede ser empleada como un generador de teoremas o un sistema experto.

En realidad se trata de un proyecto de la Universidad de Carnegie Mellon (Pittsburgh, USA), liderado por Jaime G. Carbonell y Manuela Veloso, basado en un solucionador de problemas de propósito general y, por lo tanto, independiente del dominio. Es decir, que resolverá problemas en tantos mundos (contextos) como le planteemos.

Prodigy está escrito en Common Lisp, con un código lo más independiente del sistema posible, para que pueda funcionar tanto en distintos intérpretes de Common Lisp (desde el GNU que suele venir en la distribución Slackware de Linux al CMU Clisp o el

Allegro Clisp) como en distintos sistemas operativos y máquinas.

Los mundos y los problemas se describen en PDL 4.0 (*Prodigy Description Language*), que aunque se trata de un subconjunto de LISP no nos obligará a aprender este lenguaje para poder manejar Prodigy. Bastará con conocer la sencilla sintaxis de PDL.

Prodigy es un solucionador de problemas de propósito general y, por lo tanto, independiente del dominio

EL PROBLEMA

Si algo se espera de la I.A. es que nos resuelva problemas que hasta ahora no se han podido resolver con la programación tradicional, o resultaría demasiado costoso proponérselo.

La pretensión de la I.A. es que la máquina lo haga todo a partir del momento en que el usuario le plantee el problema

Se entiende, formalmente, que se tiene un problema cuando se da la conjunción de los siguientes puntos:

1) Se quiere conseguir un objetivo.

2) No se conocen los pasos que se deben seguir para alcanzar el objetivo.

3) En la búsqueda de la solución deben existir al menos dos soluciones al problema, distintas y con distinta bondad de resultados sobre los objetivos inicialmente propuestos (es decir, habrá soluciones de distinta calidad).

Hasta ahora, con la programación tradicional, la máquina sólo hacía la última parte: la aplicación del algoritmo de cálculo o de gestión sobre los datos. La pretensión de la I.A. es que la máquina lo haga todo a partir del momento en que el usuario le plantee el problema.

Pero estos problemas pueden clasificarse dentro de varios grupos. Debemos estar abiertos a la posibilidad de que un informático encuentre soluciones a problemas distintos de los puramente numéricos o de gestión a los que estamos acostumbrados. La Inteligencia Artificial nos permitirá dar solución a problemas de sentido común (tengo que vestirme, qué me pongo si llueve, debo coger el paraguas o no, incluso problemas de moral o ética), problemas de demostración de conjeturas (basados en la demostración de teoremas y formulación lógica, consiguiendo que la máquina genere sus propias deducciones a partir de los axiomas propuestos) y problemas de juegos (compitiendo con un contrincante humano u otra máquina, aplicando funciones de evaluación para determinar cuál de los posibles movimientos me proporcionan mayor probabilidad de conseguir la victoria), etc...

CONCLUSIÓN

Una vez aclarado el concepto de Inteligencia Artificial, en el próximo número se tratarán las aplicaciones de la misma en el mundo real, en empresas, universidades y centros de investigación en general. Además, se pretenderá ofrecer al lector una visión global sobre los temas por los que podrá navegar esta serie de artículos.

Consultas al autor: carballo@lander.es

BIBLIOGRAFÍA

Referencias bibliográficas:

[Nilson, 1980] : N.J. Nilsson. Principles of Artificial Intelligence. Tioga Publishing Co., Palo Alto, California, 1980

[Rich and Knight, 1991] : E. Rich and K. Knight. Artificial Intelligence. Mc Graw Hill, New York, segunda edición, 1991.

[D. Borrajo, 1993] : D. Borrajo y otros. Inteligencia Artificial. Métodos y Técnicas. Centro de Estudios Ramón Areces. Madrid. 1993.

UTILIZACIÓN DE LOS SERVICIOS BÁSICOS CON TAPI

José C. Remiro

En el número anterior se explicaron algunos conceptos básicos de TAPI: qué es, tipos de dispositivos que distingue y qué servicios pone a nuestra disposición. En esta nueva entrega se verá la importancia de dominar las funciones *callback*.

En el artículo del mes pasado se realizó una descripción general de las capacidades de la interfaz telefónica TAPI, así como de las distintas funciones o servicios que pone a disposición del programador. Tambien se estudió con detenimiento el proceso que el sistema utiliza para notificar a la aplicación los sucesos que se producen en los periféricos asociados a los proveedores de servicios instalados en el ordenador. Para refrescar las ideas o para aquellos que no leyeron el artículo anterior, se expone a continuación un resumen a modo de recordatorio.

TAPI clasifica los dispositivos en dos tipos, de línea y de teléfono. Los dispositivos de línea son aquellos sobre los que se tiene control de alguna de sus capacidades. Los dispositivos telefónicos son aquellos sobre los que se tiene control total sobre su funcionalidad.

Los servicios TAPI se dividen en tres categorías. Los servicios básicos están soportados por todos los dispositivos. Los servicios suplementarios están soportados por dispositivos especiales. Los servicios extendidos dan soporte a funcionalidades específicas de los controladores de dispositivos.

TAPI utiliza dos tipos de nomenclatura para los números de teléfonos. La *canonical address* y la *dialable address*.

Existen dos modos de operación. En el modo síncrono, la llamada a la función retorna cuando se termina el comando. En el modo asíncrono, la llamada a la función retorna sin esperar a la terminación de la ejecución del comando. En este último caso se utilizan funciones *callback* para notificar el final de la operación. En las aplicaciones que utilicen llamadas asíncronas hay que desarrollar el bucle de mensajes de la aplicación con especial cuidado, dada la relación existente entre el correcto tratamiento de los mensajes y las llamadas a las funciones *callback*.

TAPI utiliza como parámetros en sus funciones estructuras de datos de longitud variable.

El modelo de programación para establecer una conexión se divide en registrar la aplicación, mirar las capacidades del dispositivo, negociar la versión con la que se trabaja, abrir el dispositivo de línea, realizar la conexión, enviar o recibir paquetes de datos o realizar otro tipo de operaciones y, finalmente, terminar la conexión cerrando la línea y comunicando a TAPI la finalización de la sesión.

EL EJEMPLO DE APLICACIÓN

En este artículo se pondrán en práctica los conceptos tratados hasta ahora mediante un ejemplo sencillo pero muy ilustrativo a la vez, explicando paso por paso las operaciones que hay que seguir para lograr establecer con éxito una comunicación. Se verá tambien la importancia que tiene en TAPI dominar la técnica de las funciones *callback*, así como tener un bucle de mensajes que permita a la aplicación ser notificada de los distintos sucesos relacionados con la comunicación.

La utilidad a desarrollar consistirá en un marcador telefónico auxiliado por una agenda que permite almacenar un total de 10 números de teléfono. La línea telefónica es del tipo convencional, siendo su transmisión analógica. En el CD-ROM de este mes viene el código fuente y el correspondiente ejecutable para no tener que compilarlo. La aplicación sólo se ejecuta en Windows 95 o NT, requiriendo pocas modificaciones si se quiere implementar para Windows 3.X. ha sido desa-

rrollada en C tradicional, dada la cantidad de programadores que todavía no se han decidido a dar el salto a C++. De todas maneras, encapsular las funciones y las variables que intervienen en los distintos procesos para formar una clase C++ para manejo de dispositivos es una tarea sencilla de realizar.

OBSERVACIONES PRELIMINARES

TAPI necesita seguir un orden determinado en los pasos a realizar para establecer la comunicación. Dada la naturaleza de Windows en lo que respecta a la interactuación imprevisible del usuario con los distintos elementos del interfaz, es labor del programador el impedir que el usuario ejecute los comandos de manera desordenada. Por ejemplo, habrá que evitar que se pueda efectuar una llamada sin antes haber seleccionado el dispositivo para realizarla. Lo más adecuado es deshabilitar las opciones de menú y controles para que no puedan ser utilizados, habilitándolos según TAPI va notificando a la aplicación la correcta ejecución de los distintos comandos TAPI.

VALORES DE RETORNO

Todas las funciones TAPI devuelven uno o más resultados en el valor retornado por la función y en las direcciones de las variables indicadas como parte del argumento de la función. En el resultado devuelto como valor de retorno de la función hay que distinguir los tres tipos siguientes:

Si el resultado es *0*, significa que la función se ha ejecutado de forma síncrona, pudiendo utilizar cualquier otro resultado opcional devuelto por TAPI de manera inmediata. Un valor de retorno positivo indica que la función se ejecuta en modo asíncrono. Cuando se recibe el mensaje de notificaión de terminación de comando, si éste es satisfactorio, indica que cualquier valor escrito como resultado de la llamada a la aplicación es válido y se puede utilizar. Mientras no se reciba el mensaje de notificación, estos valores no deben considerarse como válidos. El orden de recepción de los mensajes de notificación no tiene por qué coincidir con el de ejecución de los comandos, debido a que hay comandos asíncronos que tardan más en ejecutarse que otros.

CUADRO 1

LINEBEARERMODE_VOICE	Es la opción por defecto e indica una transmisión analógica a 3.1 KHz. Se utiliza para líneas analógicas convencionales y soporta la transmisión de voz, fax y datos.
LINEBEARERMODE_SPEECH	Corresponde a la norma G.711 para transmisión de conversaciónes a través de redes locales. Se pueden utilizar algoritmos de compresión y descompresión para reducir la carga. No está orientado a soportar fax y módem.
LINEBEARERMODE_MULTIUSE	Se utiliza en comunicaciones RDSI.
LINEBEARERMODE_DATA	Para restringir la transmisión a datos.
LINEBEARERMODE_ALTSPEECHDATA	Permite alternar entre la transferencia de datos y conversación sobre la red..
LINEBEARERMODE_NONCALLSIGNALING	No se asocia ningún tipo de comunicación al dispositivo.
LINEBEARERMODE_PASSTHROUGH	Da el control del tipo de acceso directamente a la aplicación, no siendo el proveedor de servicios quien pone los límites.

Por último, un valor negativo indica que ha ocurrido algún error.

INICIO DE LA SESIÓN

Antes de poder realizar operación ninguna, hay que hacer saber a TAPI que se van a utilizar sus servicios para que éste los ponga a disposición de la aplicación. La mejor manera de realizarlo es durante la inicialización de la aplicación. Para ello, se llama desde la rutina encargada de inicializar la instancia de la aplicación a la función *lineInitialize*, pasando como parámetros un puntero a una estructura del tipo *HLINEAPP* que TAPI rellena con un *handle* que asocia a la aplicación para su uso con el resto de funciones, la instancia de la aplicación, la direccioón de la función *callback* que se utilizará para recibir los mensajes de notificación, una cadena ASCII con el nombre que queremos asociar a la aplicación de cara a que otras aplicaciones la identifiquen cuando se realice o reciba una llamada en el sistema, y la dirección de una variable que al retornar de la función indica el número de dispositivos disponibles. Esta última variable se utilizará más adelante para averiguar las características de los dispositivos conectados.

SELECCIÓN DEL DISPOSITIVO

Tanto para poder realizar llamadas como para recibirlas, hace falta contar con un dispositivo de línea que permita realizar tales operaciones. Como puede haber perfectamente más de un dispositivo conectado, en la aplicación se permite al usuario, mediante la opción del menú principal *Configuración*, escoger el que quiera de los listados en el *ListBox*. El programa llama a la función *lineNegotiate ApiVersion* para negociar la versión TAPI a utilizar entre la aplicación y cada dispositivo a examinar. Los parámetros que se pasan son el número de dispositivo, la versión mínima y máxima para que la aplicación pueda funcionar, un puntero a la variable que retornará la versión a utilizar y un puntero a una estructura para almacenar identificadores para los servicios extendidos del proveedor en caso de tenerlos. El segundo paso consiste en llamar a la función *lineGetDevCaps* para averiguar las características del dispositivo y obtener un nombre descriptivo que se utilizará para visualizarlo en la lista de dispositivos a escoger. Nótese que hay que llamar dos veces a la función. En la primera llamada se pasa la estructura *LINEDEVCAPS* con su tamaño básico indicado en el campo *dwTotalSize*, devolviendo a la salida el tamaño requerido en el campo *dwNeededSize*. Acto seguido se redimensiona dinámicamente el tamaño de la estructura y se vuelve a llamar a la función con el tamaño adecuado indicado en *dwTotalSize*. Los parámetros que se pasan a esta función son el número de dispositivo, la versión API, la versión extendida si se obtuvo con *lineNegotiate ExtVersion*, y un puntero a una estructu-

ra con una cantidad de campos bastante extensa en la que se detallan todas las características. Antes de salir de la ventana de configuración se mira si se seleccionó algún dispositivo y si hay seleccionado ya algún número de la agenda, activando la opción de menú *Llamar* en caso afirmativo.

PARÁMETROS DE COMUNICACIÓN

Dos datos a tener muy en cuenta cuando se realiza una conexión son los parámetros *LINEBEARERMODE* y *LINEMEDIAMODE*.

INEBEARERMODE indica la calidad de la comunicación, siendo la opción por defecto la comunicación a 3.1 KHz sobre una línea analógica convencional, soportando voz, datos, conversación y fax.

LINEMEDIAMODE especifica el tipo de comunicación a realizar, existiendo cuatro tipos básicos que son: voz, datos, fax y conversación.

Estos dos datos se pasan a la función encargada de abrir la línea, y se puede saber si el tipo de comunicación a realizar es soportado por el dispositivo examinando los campos *dwBearerModes* y *dwMediaModes* de la estructura devuelta por *lineGetDevCaps*.

OBTENCIÓN DE LA LÍNEA

Cuando una aplicación se dispone a tomar la posesión de una llamada, tiene que indicarle a TAPI el tipo de privilegio que va a tener sobre la misma. Esto se hace especificándoselo como un parámetro de *lineOpen*. El privilegio con el cual una línea es abierta permanece en efecto para subsecuentes llamadas usadas en la aplicación sobre esa línea. Una aplicación siempre es duaña de los privilegios al crear una llamada. Cuando la aplicación abre una línea para colocar una llamada invoca a *lineOpen* con el privilegio *LINECALLPRIVILEGE_NONE*, el cual aisla a la aplicación de llamadas entrantes mientras permite realizar llamadas al exterior.

REALIZANDO LA LLAMADA

La aplicación realiza la llamada con *lineMakeCall*, indicando el número de teléfono y el tipo de llamada a realizar (en este caso de voz) pasando el modo de comunicación *LINEMEDIAMODE_INTERACTIVEVOICE* para indicar que se trata de una llamada de voz y el modo *LINEBEAREMODE_VOICE* para indicar que se quiere una calidad de transmisión de 3.1 KHz, el cual soporta voz, fax y datos sobre una línea analógica (ver tabla 1). Al ser una función asíncrona, retorna un valor positivo en caso de iniciarse el comando de manera satisfactoria, o un valor negativo si ocurre algún error, por ejemplo, que la línea esté ya en uso (*LINEERR_CALLUNAVAIL*). Cuando la función *lineMakeCall* se ha completado satisfactoriamente, la aplicación recive el mensaje *LINE_REPLY*, indicando que el *handle* retornado por la función *lineMakeCall* es válido. A partir de ese momento, si se quisiese realizar una transferencia de datos, se utilizaría el *handle* para sobrecargar las funciones estándar de entrada/salida y poder realizar el intercambio de información. Dicho método se tratará en un capítulo posterior cuando se hayan explicado los conceptos de multitarea necesarios para mantener una transferencia asíncrona. Si se indica *NULL*, indica el valor por defecto de 3.1 KHz, el cual sirve para soportar voz, modem y fax (TAPI no debe de utilizarse para la transmisión de faxes, debiéndose utilizar en su lugar MAPI.) Cuando la llamda es establecida pasa por diferentes estados, cada uno de ellos resulta en un mensaje enviado a la aplicación del tipo *LINE_CALLSTATE*. Estos estados pasan por *dialtone* (tono de marcado), *dialing* (marcando), *ringback* (esperando contestación), y si la conexión es establecida, conectado (*LINECALLSTATE_CONECTED*). Después de haber recibido el mensaje de conectado, la aplicación puede comenzar a enviar datos.

CUADRO 2

LINEMEDIAMODE_UNKNOWN	Especifica que se desconoce el tipo de comunicación que se está realizando.
LINEMEDIAMODE_INTERACTIVEVOICE	Se utiliza para llamadas de voz habiendo personas hablando a ambos extremos de la línea.
LINEMEDIAMODE_AUTOMATEDVOICE	La aplicación responde de manera automática a la fuente de voz del otro extremo de la línea.
LINEMEDIAMODE_DATAMODEM	Indica transmisión de datos vía módem.
LINEMEDIAMODE_G3FAX	Para fax del grupo G3.
LINEMEDIAMODE_G4FAX	Para fax del grupo G4.
LINEMEDIAMODE_DIGITALDATA	Para transmisión de datos digitales.
LINEMEDIAMODE_TELETEX	Servicios de teletex.
LINEMEDIAMODE_VIDEOTEX	Servicios de videotext.
LINEMEDIAMODE_TELEX	Servicios de télex.
LINEMEDIAMODE_MIXED	Servicios MIXED soportados por RDSI.
LINEMEDIAMODE_ADSI	Transmisión de imágenes analógicas.

LA FUNCIÓN *CALLBACK*

Como se ha indicado en varias ocasiones, el mecanismo utilizado por TAPI para notificar la terminación de los comandos asíncrono y otro tipo de sucesos es mediante la función *callback*. Esta función recibe la notificación de sucesos en forma de mensajes, por un estilo a como se reciben en las funciones de procedimientos de ventana. Junto con el mensaje se reciben los siguientes parámetros: el identificador del dispositivo que realiza la notificación, para que pueda ser localizado por la aplicación; el mensaje propiamente dicho; la instancia de la aplicación; y tres parámetros que varían su significado dependiendo del tipo de suceso notificado. Todas las funciones asíncronas notifican su terminación mediante un mensaje del tipo *LINE_REPLY* o *PHONE_REPLY*.

FINALIZACIÓN DE LA LLAMADA

Un mensaje del tipo *LINE_CALLSTATE* informa a la aplicación de que la llamada ha finalizado. Cuando se desea dar por finalizada una llamada sin que en el otro extremo se haya "colgado" se utiliza la función *lineDrop*. Una vez hecho esto, hay que liberar el *handle* asignado para la transmisión de información mediante la función *lineDeallocateCall*..

DISTRIBUCIONES DE LINUX: INSTALACIÓN

César Sánchez

Linux ha conseguido en muy poco tiempo alcanzar una alta cota de popularidad, especialmente en el entorno de Internet y entre los amantes de la programción. Su licencia, como más adelante se comenta, permite que varias compañías puedan competir a la hora de hacer colecciones de software más completas o de mayor calidad basadas en Linux. Es lo que se conocen como distribuciones.

DISTRIBUCIONES

Una distribución de Linux es un conjunto de software que contiene un kernel (ver glosario), aplicaciones y programas de utilidades, y algún método de instalación. Aunque todas las distribuciones se basan en el mismo kernel y en las mismas aplicaciones, se diferencian porque presentan alguna característica peculiar. Así, no se puede hablar en general de la mejor distribución, sino más bien de distribuciones que encajan mejor a diferentes usuarios. En el presente artículo se hará especial énfasis en la distribución Debian, aunque también existen algunas otras, entre las que cabe citar Red Hat, Slackware, Linux FT, Caldera o Yggdrasil. También existen otras compañías que se dedican a recopilar varias de estas distribuciones y cantidades de otros programas de forma más heterogénea, a través de servidores de ftp en Internet, y venderlos en formato de hasta 6 o 7 CD-ROMs, por precios en ocasiones muy competitivos. Son el caso de InfoMagic (cuya portada puede verse en la figura 1) o la propia Yggdrasil.

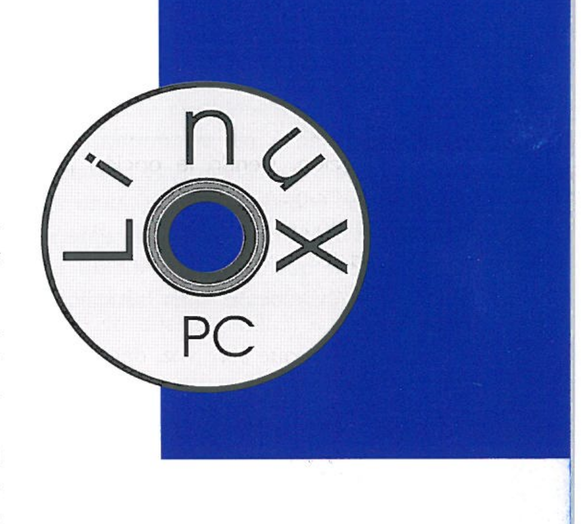
Una distribución convencional puede ocupar fácilmente cientos de Megabytes, ya que contiene multitud de programas y utilidades para los más diversos campos. Muchos de los programas que se incluyen en las distribuciones pertenecen al proyecto GNU, por lo que en ocasiones se conoce al sistema operativo en su totalidad como GNU/Linux o "sistema operativo GNU basado en Linux". El propio kernel del sistema operativo (Linux propiamente dicho) está registrado, protegido por copyright bajo la licencia GPL (*GNU General Public License*).

Una confusión bastante común cuando se habla de distribuciones suele venir desde el punto de vista de la numeración de las versiones. La numeración de éstas no coincide con las del kernel del sistema operativo. En el caso del kernel, el sistema de numeración que se sigue es el siguiente:

El identificador de la versión consta de tres números. El primer número es el llamado "*major version number*", el segundo "*minor version number*" y el tercero "*patch level*". Números pares en el *minor version number* indican versión estable, kernel de calidad de producción, comercial. En cambio, números impares indican que el kernel es experimental, con todas las nuevas características recien implementadas. En el momento de la entrega de este artículo, el kernel más actualizado era el 2.0.26 en la serie estable y 2.1.13 en la inestable.

En el caso de las distribuciones la numeración es, de alguna manera, más libre. Por citar algún ejemplo, en el momento de la publicación la última versión de Slackware era la 3.1, y de Debian la 1.1.14.

Las distribuciones, en general, están divididas en conjuntos más pequeños y homogeneos de software, que con frecuencia se denominan "paquetes". Un paquete puede contener, por ejemplo, un compilador de C o una librería de algoritmos genéticos, tipos de letras para el sistema de ventanas XWindow,



Tras un breve paréntesis, vuelve a nuestras páginas la sección dedicada al popular Linux. En esta sección, Foro Linux, diversos colaboradores intentaran dar a conocer a los no iniciados este sistema operativo.

o el popular juego Quake. Los paquetes también siguen, por lo general, una numeración diferente. No es necesario conocer la versión de cada uno de los paquetes que se desea usar, ya que las diferentes distribuciones se encargan de homogeneizar introduciendo, con diferentes criterios, una elección entre estabilidad y novedad.

Las diferencias entre las varias distribuciones radican, fundamentalmente, no en contener paquetes que sean diferentes, sino en que la compilación de éstos se ha llevado a cabo de una manera distinta; o en que presentan la última versión del paquete o, por el contrario, tan sólo software exhaustivamente probado, con afán de conseguir un producto de alta calidad comercial. O simplemente se distinguen en que tratan de incluir un método de instalación más sencillo o agradable.

DEBIAN GNU/LINUX

Una característica de la distrución de GNU/Linux llamada Debian, y que le hace ser especialmente atractiva, es el sistema de mantenimiento de paquetes. Este consiste en considerar las dependencias que puedan tener unos paquetes con otros. Si un paquete contiene una librería gráfica con el código necesario para la ejecución de un programa de procesamiento de imagen, esto es indicado en el paquete de dicho programa, y así éste no podrá ser instalado si la librería no lo es.

Adicionalemte, realizar una actualización a una versión superior de alguno de los componentes no convierte progresivamente al sistema en algo más grande, con pedazos obsoletos esparcidos por el disco duro y archivos de configuración cada vez más voluminosos. La antigua versión es retirada con todos sus archivos (siempre y cuando ningún otro paquete utilice alguno de éstos, claro está) y luego se desempaqueta la nueva. Además, la información para configuración de los paquetes recién instalados, para que se adapten a la idiosincrasia de la máquina, se incluye también en el propio paquete, de tal manera que ésta puede ser realizada del modo más automático posible.

INSTALACIÓN DE DEBIAN

Para instalar Debian hay que pensar previamente en los requisitos que la máquina debe cumplir . El sistema mínimo consiste en un 386 con 4 Mb de RAM, y aunque se insiste en que con 2 Mb se puede hacer arrancar Linux, el funcionamiento no sería demasiado bueno en cuanto a prestaciones. Un sistema al que se le emepezaría a sacar verdaderamente partido sería un 486 con 8 Mb de RAM. No cabe hablar del sistema óptimo, ya que Linux aprovechará los recursos de la computadora al máximo, y así cuanta más memoria o CPU mejor. Aún así, con un Pentium con 32Mb de RAM y una tarjeta de video con 2 Mb, Linux puede alcanzar unas prestaciones realmente envidiables. Debe reseñarse que Linux es de los pocos sistemas, debido a la disponibilidad de su codigo fuente, que puede aprovechar al máximo las características del procesador. Como se vera más adelante, se puede optimizar el código para Pentium o incluso Pentium Pro (en otros sistemas con canales de distribución más convencionales, el código instalado será el mismo tanto si disponemos de un 386 como un Pentium Pro).

En cuanto a espacio de almacenamiento, aunque Linux puede funcionar incluso en un disquete, para aprovechar el gran número de herramientas que se presentan en la distribución se deberá reservar en el disco duro un espacio de 60Mb como mínimo (300 Mb suele ser una cantidad suficiente para una instalación medianamente completa y con un espacio de trabajo adecuado).

Cabe indicarse también que no se incluyen en el CD-ROM de la revista distribuciones de Linux para estaciones Sun SPARC, Alpha o 68k debido a problemas de espacio, ya que aunque Linux funciona para estas arquitecturas, el PC con procesador Intel o compatible es una plataforma mucho más popular.

Se describen a continuación los pasos para llevar a cabo una instalación de Debian GNU/Linux, que aunque no es la que ofrece un programa de instalación más sencillo o amigable presenta las ventajas antes mencionadas.

Debe leerse, en primer lugar, el fichero README que aparece en la base de la distribución, que se encuentra en el directorio \DEBIAN del CD. Una guía para la instalación completa y una descripción detallada del hardware soportado puede encontrarse en el archivo install.txt en el directorio /buzz-updates/disks-i386/current/ .

El primer paso será volcar los disketes de arranque para comenzar la instalación. Los archivos necesarios se encuentran el directorio anteriormente indicado. En el caso de tener una unidad de arranque 3 y 1/2, los archivos a volcar son boot1440.bin , root.bin, base14-1.bin, base14-2.bin y base14-3.bin, es decir, 5 disquetes en total. En el caso de disponer de unidad de 5 y 1/4 serán boo1220.bin, root.bin, base12-1.bin, base12-2.bin, base12-3.bin, base12-4.bin. Para volcar estos programas a los disquetes no basta con copiarlos a un disquete formateado con Ms-DOS o Windows95, ya que en realidad contienen información acerca de lo que debe ir en cada sector del disquete, es decir, son "imágenes" del mismo. Para realizar el volcado de la manera adecuada se deberá utilizar el programa RAWRITE2.EXE, que puede encontrarse en el directorio \TOOLS de la distribución. Para obtener instrucciones más detalladas de cómo usarlo puede leerse el archivo RAWRITE.TXT.

Linux necesita una partición de disco para instalarse, es decir, que su instalación no puede realizarse en el espacio libre de un disco duro Ms-DOS. Si es esto de lo que se dispone puede, sin perderse la información, reducirse el tamaño de esta parción para dejar sitio usando el programa FIPS. Éste puede encontrarse también en el directorio \TOOLS, con información propia. Los pasos son muy simples: primero se debe "defragmentar" el disco, por ejemplo con el programa DEFRAG disponible en Ms-DOS desde la versión 6.0, resultando toda la información almacenada en el disco en los sectores iniciales. FIPS se encargará de rehacer la tabla de particiones situando el nuevo límite por encima de los sectores más altos ocupados.

Una vez que se tiene un disco entero, o espacio para una partición, se procede a arrancar con el disquete *boot*. Salvo que se disponga de un sistema con una tarjeta de red o dispositivos SCSI, no demasiado convencionales, bastará con pulsar <ENTER> tras la primera pantalla, según se indica. Esto arrancará por primera vez el sistema operativo Linux. Una vez cargado el kernel, éste pedirá la introducción del segundo disquete, el *root*, que contiene un conjunto mínimo de pro-

gramas básicos para continuar con la instalación.

En este punto comienza la instalación propiamente dicha. El primer paso consiste en particionar el disco duro. Típicamente se necesitarán dos particiones, una para volcar el sistema y otra para *swap* (ver glosario). En sistemas más potentes, con muchos usuarios y aplicaciones más críticas (por ejemplo, funcionando como servidores de Internet) convendrá realizar más particiones, para diferentes partes del árbol de directorios.

Con respecto al tamaño a asignar para cada partición, una regla sencilla suele ser asignar el doble de memoria para *swap* que la memoria RAM disponible (hasta un máximo de 32 Mb) reservando el resto para la instalación de programas. La partición para la instalación deberá etiquetarse como tipo *Linux native* (por defecto se hace así) y la de *swap* como tipo *Linux swap*.

Debe notarse que la manera de nombrar los discos en Linux es diferente que en Ms-DOS, y de alguna forma más canónica. En DOS se nombra la primera partición del primer disco como C: y se va avanzando letras según se recorren las particiones del primer disco, luego el segundo disco, etc... Así, si se dispone de un disco y un CD-ROM, éste, típicamente se denotará con la letra D. En cambio, si el primer disco tuviese dos particiones (para tener Ms-DOS y OS/DOS o Win95, por ejemplo) éstas serían C: y D:, y el CD-ROM, siendo el mismo, ahora se denotaría como E. Sólo por el hecho de modificar un disco se ve afectado el nombre de otro. En Linux, sin embargo, el primer disco es */dev/hda*, el segundo */dev/hdb*, el tercero */dev/hdc* y el cuarto */dev/hdd*. Las particiones se denotan con números tras el nombre de su disco (*/dev/hda1*, */dev/hda2* ...) independientemente de los demás discos.

Tras particionar el disco se nos preguntará qué particiones deberán utilizarse para *swap* y para la instalación, pero el programa averiguará las posiblidades y, normalmente, la respuesta por defecto será la más normal (si no la única).

Una vez resuelto satisfactoriamente el paso de las particiones, pedirá que instalemos un sistema básico, que consiste en unos pocos binarios, más de los que caben en un simple disquete. Para ello sirvió volcar los disquetes marcados como base. El programa irá pidiéndolos uno a uno en la disquetera.

Tras ello, será pedido un disquete con un kernel para poder incluirlo en este minisistema en disco duro, y poder arrancar sin floppies. Valdrá con introducir el disquette de *boot* (el primero que se utiliza para arrancar). El programa de instalación pedirá ahora si queremos introducir módulos (ver glosario) en el kernel, con unos sencillos menús. Los módulos son trozos del kernel (que puede ser código necesario para utilizar algo de hardware, como una disquetera o un puerto serie o paralelo, o para poder leer sistemas de ficheros de multitud de sistemas operativos como Windows95 o OS2). Se recomienda elegir únicamente los puertos serie y paralelo y, si se dispone de CD-ROM que no sea IDE-ATAPI, el soporte para ese CD-ROM. Si la instalación va a realizarse desde CD será necesario instalar el módulo *isofs*.

Se pasa a continuación a la etapa de configuración del sistema básico instalado en el disco duro. Se podrá elegir el tipo de teclado del que se dispone (típicamente responderemos *es.map*, es decir, el teclado español). También habrá que elegir la zona horaria en la que nos encontramos para hacer ajustes autamáticos de cambio de hora, etc... Esta pregunta se responde en función del continente y ciudad, donde estemos, entre las posiblidades que se nos ofrece. Una respuesta típica puede ser Europe/Madrid.

El paso siguiente es configurar la red. Unix, en general, y Linux en particular es un sistema muy bien adaptado a redes de comunicaciones como Internet. Es por ello por lo que goza de una mayor aceptación y popularidad dentro de la comunidad Internet que otros sistemas más convencionales o que gozan de mayor publicidad. Se preguntará por el nombre que debe tener la máquina, y si tras ello se responde que se está conectado a alguna red, pedirá el nombre del dominio y la dirección de Internet y otros datos técnicos. En el caso de estar, efectivamente, conectado, el administrador de la red los habrá proporcionado (datos como *gateways*, servidores de nombres...). Tras ello no será obligatorio rearrancar la máquina, y las tablas y nombres pertinentes serán automáticamente configurados.

El siguiente paso consiste en instalar un pequeño programa llamado LILO (*LInux LOader*) que permite, al arrancar la máquina, elegir entre los diferentes sistemas operativos instalados. Linux no evita que se puedan usar otros sistemas, y no presupone que el disco duro esté exclusivamente destinado para su instalación. Con LILO se podrá arrancar Ms-DOS, OS/2, Linux o cualquier otro sistema que esté instalado en particiones o discos diferentes.

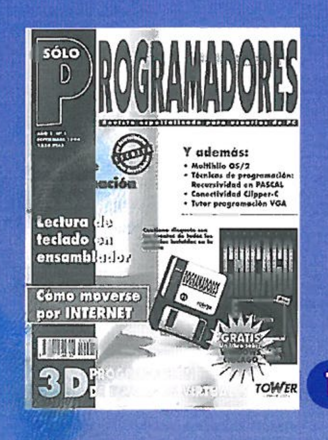
Tras este último paso se está preparado para arrancar directamente desde el disco duro. Se ofrecerá la opción de crear un disquete especial para arrancar con él si algo fallase. Crearlo es muy recomendable.

Tras arrancar la máquina y ejecutar el kernel de Linux, se pedirá cambiar la *password* (ver glosario) de *root*. El *login root* es el que usa para el administrador. Este usuario tendra permisos especiales para hacer cosas especiales. Por ejemplo, es el único que puede instalar programas nuevos para que todos los demás usuarios puedan usarlos o tocar configuraciones globales de algunos programas (aunque muchos de ellos permiten configuraciones personalizadas y diferentes para cada usuario). La *password* de *root* debe ser conocida, de entre todos los usuarios de la máquina, sólo por aquellos que vayan a realizar tareas administrativas. También se insta a que se cree una primera cuenta para otro usuario y que se trabaje (salvo en las tareas de administración) con ella. Un nombre de *login* puede ser puede ser nuestro propio nombre "*cesar*". Otro método muy utilizado en Internet consiste en utilizar las primeras letras del nombre y los apellidos, como en "*css*".

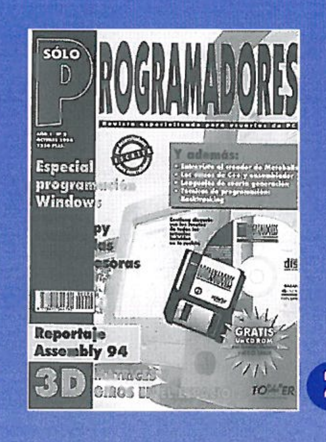
Tras las cuentas, se entra en el programa de selección de paquetes a instalar, llamado *dselect*. Primero se le deberá decir qué "*Media*" se va a utilizar para leer los paquetes en la instalación. En nuestro caso será desde CD-ROM, aunque también se permite realizarla desde *ftp* por Internet o a través de otra máquina en la misma red de área local,... La única información que será necesario aportar es el disco que corresponde al CD-ROM en el formato arriba indicado.

(continuará sección CD-ROM)

¡Que no te falte ni uno!



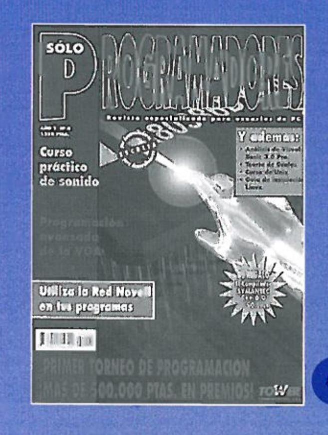
1



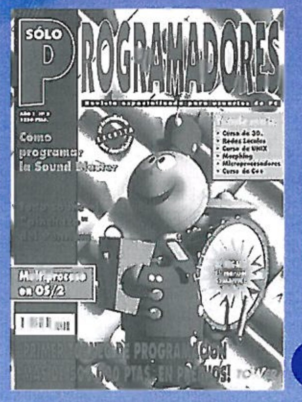
2



3



4



5



6



7

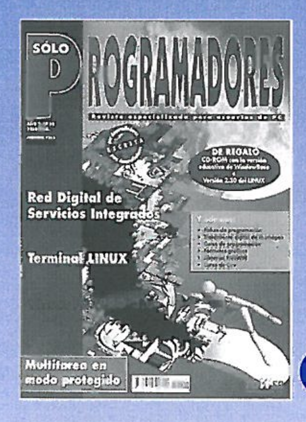


8

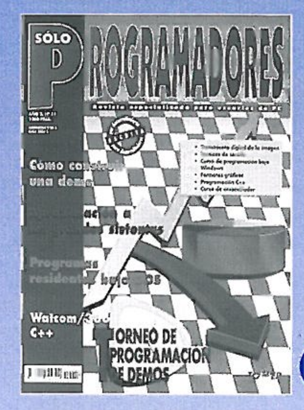




9



10



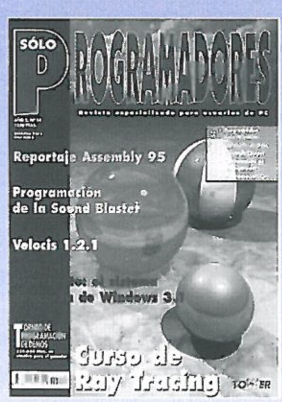
11



12



13



14

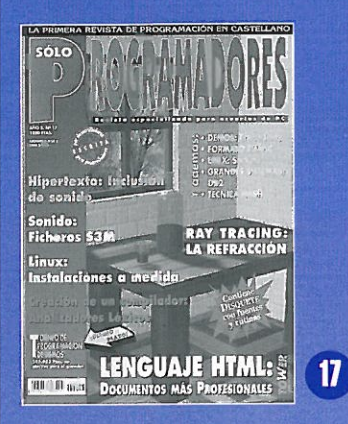


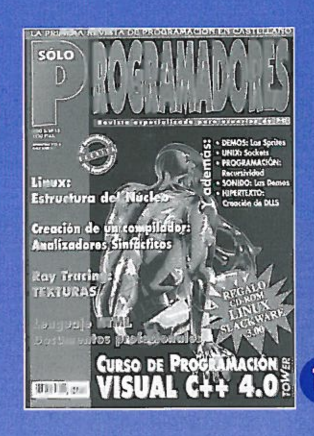
15

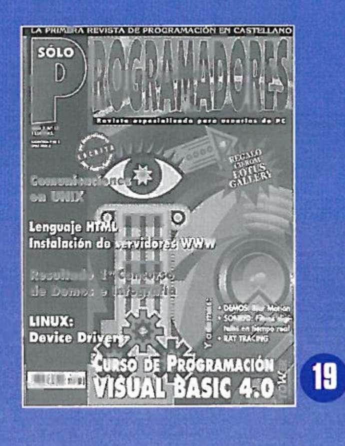


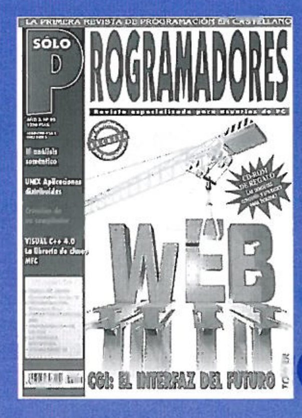


·NÚMEROS ATRASADOS·NÚMEROS ATRASADOS·NÚMEROS ATRASADOS·

























COMPLETA YA TU COLECCIÓN

Solicite los números que le falten enviando este cupón por correo o fax (91) 661 43 86, o llamando al teléfono (91) 661 42 11*.
Horario lunes a jueves de 9 a 14 y 15 a 18h. y viernes de 9 a 15h.

SOLICITO LOS SIGUIENTES NÚMEROS ATRASADOS DE SÓLO PROGRAMADORES:

☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9 ☐10 ☐11 ☐12 ☐13 ☐14 ☐15 ☐16 ☐17 ☐18 ☐19 ☐20 ☐21 ☐22 ☐23 ☐24 ☐25 ☐26 ☐27 Importe..(1.250 pts. cada uno)

Nombre y apellidos...Domicilio...

Población...C.P ....Provincia.........................Telf...............................Profesión....................

FORMA DE PAGO:

☐ Con cargo a mi tarjeta VISA nº ____ ____ ____ ____

Fecha de caducidad de la tarjeta................Nombre del titular, si es distinto...

☐ Contra-reembolso del importe más gastos de envío.

☐ Cheque a nombre de TOWER COMMUNICATIONS S.R.L., que adjunto.

☐ Giro Postal (adjunto fotocopia del resguardo).

Firma:

Rellena este cupón y envíalo a:
TOWER COMMUNICATIONS SRL
C/ Aragoneses, 7
28100 Pol. Ind. ALCOBENDAS (Madrid)

CONTENIDO DEL CD-ROM

DEBIAN GNU/LINUX, TCL/TK, GNAT ADA, GUÍA-SP ...

Continua sección Foro Linux

Se comenta aquí, a modo de referencia, una pequeña parte de los paquetes que pueden encontrarse en Debian GNU/Linux.

HERRAMIENTAS DE ADMINISTRACIÓN

hdparm: Permite modificar los parámetros de la controladora y del disco y conseguir mejores prestaciones.
quota: Para establecer restricciones al espacio de disco utilizado por cada uno de los usuarios.
cron: Permite realizar tareas periódicamente.

PROGRAMACIÓN

En lo relativo a lenguajes se puede encontrar:
gcc: GNU C Compiler. Es un compilador altamente portable (existen versiones para gran cantidad de arquitecturas, incluido Ms-DOS) y que permite construir versiones para múltiples lenguajes. Por ejemplo, *gnat*, ha sido construido partiendo de *gcc*. También se presenta una versión para la generación de código win32.(lo que permite desarrollar código para Windows usando Linux como plataforma de desarrollo).
g++: Compilador de C++ de gnu basado en gcc.
gnat: *GNU and NYU Ada Translator* en su versión 3.0.4. Es un compilador de Ada basado en gcc.
g77, for77 y ratfor77: Compiladores de Fortran77. También se dispone de *f2c*, que es un conversor a C.

p2c: Conversor de Pascal a C.
gcl: *GNU common lisp*. Es un compilador de Common Lisp. Este paquete ha sido compilado con TCL/Tk para permitir diseñar interfaces de usuario para Xwindow.
guile: *GNU extension language*. Es una librería para extensión de lenguajes que incluye una máquina virtual, un sistema de ejecución e interfaces para múltiples lenguajes. En la actualidad *scheme* es el lenguaje mejor soportado por *guile*. También incluye extensiones TCL/Tk.
perl: Implementación del popular lenguaje interpretado Perl (*Larry Wall's Practical Extracting and Reporting Language*).
objectlib: En realidad, el propio *gcc* soporta el lenguaje *objective C*. Este paquete contiene una extensa librería de clases para él.
tcl: *Tool Command Language*. Es un lenguaje interpretado, potente y fácil de aprender. Junto a Tk es muy utilizado en la realización de interfaces de usuario gráficos.
python: *python* es un lenguaje orientado a objetos interpretado. Incluye una librería con gran cantidad de facilidades para administración, gráficos y sonido.
En cuanto a herramientas y librerías se encuentran:
gdb: *GNU DeBugger*. Permite depurar programas compilados.
xxgdb: Version de *gdb* para XWindow, con un interfaz gráfico más amigable.
tk: Es un *toolkit* para generación de interfaces de usuario. Aunque inicialmente estuvo implementado en Tcl, se está tratando de desarrollar como librería independiente para lenguajes compilados. La principal plataforma es XWindow, aunque actualmente también existen versiones para MsWindows.
slang, ncurses: Librería para desarrollo de interfaces de usuario en modo texto. *ncurses* proviene de la librería *curses* del Unix de BSD.
libgr: Permite desarrollo de código para manejo de gráficos en los formatos más habituales: FBM, JPEG, PBM, PGM, PNG, PNM, PPM, RLE, TIFF y ZLIB.
libpng: Librería para utilizar el formato *Portable Network Graphics* (PNG).
ilu: *InterLanguage Unification system* es un sistema con interfaz de objetos multilenguajes. ILU permite abstraer el espacio de direcciones (varios procesos pueden usar el mismo objeto), el lenguaje de programación e incluso el sistema operativo. También puede ser usado para diseñar sistemas distribuidos.
svgalib: Esta librería ofrece servicios para realizar programas gráficos en la consola del sistema en lugar de en un sistema de ventanas como Xwindow.
libjpeg: Librería del *JPEG Independent Group* para el tratamiento de imágenes en formato JPG.
mesa: Una libreria gráfica que provee el interfaz OpenGL de Silicon Graphics. Incluye programas de ejemplo.
También se incluyen herramientas para facilitar la realización de proyectos software:
make: *GNU make* permite realizar la compilación de las piezas del proyecto que hayan sido modificadas y el enlazado en el programa ejecutable de manera automática. La versión incluida es 3.75.
rcs: Provee mecanismos para realizar un control de versiones del proyecto. Con ella se puede gestionar el dar marcha atrás en cambios,...
cvs: Es un sistema de versiones que facilita el control concurrente de las mismas. Está montado sobre *rcs* y significa un paso más adelante, permitiendo que múltiples usuarios participen ordenadamente en el desarrollo.
yacc, bison, flex: Son generadores de analizadores léxicos o *parsers*. Se parte de la descripción del lenguaje a analizar y la herramienta genera un código capaz de analizar textos en ese lenguaje.

electric-fence: Provee un sistema de depuración de programas en cuanto a lo que a reserva y liberación de memoria dinámica se refiere.

COMUNICACIONES

Se incluyen herramientas para sacar provecho a la conectividad que se disponga (módem, ISDN, emisora de radioaficionado...):
efax, mgetty, xringd: Permiten recibir llamadas por módem y faxes, abre una sesión para una conexión remota o imprimir, almacenar o mostrar en pantalla en el caso de un fax.
minicom: Es un programa de comunicaciones guiado por menús, muy similar al popular Telix de MsDOS.
lrzsz: Permite enviar y recibir ficheros por el módem haciendo uso de los protocolos *zmodem, kermit, ymodem,...*
c10cfgd: Es un programa para redes de radiopaquete (usando emisoras de radioaficionado).
elm, pine: Lectores de correo electrónico.
sendmail, smail: Agentes de transporte de correo electrónico.
En cuanto a servidores:
apache: Completo servidor de Web.
ipx: Protocolo de Novell.
irc: Servidor de *Internet Chat Script*.
lpr: Servidor de impresora.
lynx: Navegador ultrarápido de WWW en modo texto.
wuftpd: Servidor de *ftp* (protocolo de transferencia de ficheros de Internet).
ytalk: *Talk* multiusuario, permite tener conversaciones electrónicas.
nntp, tin: Servidor y cliente de noticias de Internet.

UTILIDADES Y HERRAMIENTAS XWINDOW

xfig: Con esta herramienta se pueden componer gráficos y diagramas más o menos complejos e imprimrlos en postscript.
xbmbrowser: Visor de imágenes en formato bitmap y pixmap.
xcoral: Un editor que indenta el código fuente C o C++ y lo colorea, y permite navegar por las funciones o clases.
xpaint: Creador de imágenes bitmap o pixmap. Es un básico PAINT-BRUSH.
xdvi: Permite visualizar texto para impresora en formato DVI.
xloadimage: Visor de gráficos bajo XWindow. Acepta formatos NIFF, SUN-RASTER, GIF, JPEG, TIFF, FBM, CMU-RASTER, PBM, FACES, RLE, XWD, VFF, MCIDAS, VICAR, PCX, GEM, MACPAINT, XPM Y XBM, visualizandolos en una ventana o en el fondo. También permite procesado de imagen.
fvwm95: *Window manager* con el aspecto Windows95.
xbl: Un juego similar al Tetris en 3D.
xboard: Entrada gráfica al *gnuchess*. Permite visualizar la partida y jugar dos humanos, uno contra la máquina o poner a la máquina manejando los dos jugadores.
xboing: Una versión del popular Arkanoid para XWindow.

MATEMÁTICAS

gnuplot: Crea gráficos postscript a partir de tablas o fórmulas matemáticas. Dibuja curvas en 3D.
ctave: Herramienta matemática estilo MATLAB.

SONIDO

workbone: Permite escuchar CDs de audio.
sox: Conversor entre muchos formatos de sonido.
cdtools: Utilidades para clasificar Cds.
xmix: Controla gráficamente los parámetros de la tarjeta de sonido.

EDITORES

emacs: El potente y altamente configurable entorno de GNU.
vi: Editor clásico en sistemas Unix. es muy pequeño y los expertos le sacan una alta productividad, pero su curva de aprendizaje no es excesivamente buena. Se incorporan versiones mejoradas (*nvi, vim*).
jed: Editor expandible basado en Slang.

TEXTO E IMPRESORAS

ghostview: Visualizador de postscript para XWindow o SVGAlib.
genscript: Permite transformar texto ASCII en un documento postscript.
apsfilter: Este filtro de impresora da facilidades para mandar a la misma multitud de formatos.
gs: Interprete de postscript. Lo utiliza ghostview y también vale para imprimir documentos postscript en impresoras que no lo soportan.
xpdf: Visualizador del formato de Xerox PDF.
Se incorpora también el sistema de preparación de documentos LaTeX con un gran conjunto de herramientas y tipos de letra.

JUEGOS

Entre otros, se incluyen los siguientes:
abuse: Juego arcade para XWindow y SVGAlib.
maelstrom: Asteroides.
xonix: Juego de habilidad.
mirrormagic: Juego de estrategia.
Quake: Motor del nuevo juego de Id, sucesor del Doom.
fortune: "galletas de la fortuna", pequeñas frases curiosas y citas.

VARIOS

cdwrite: Permite utilizar grabadoras de CD-ROM.

dbview: Visualiza ficheros DBase.

gpm: Programa que permite cortar y pegar en modo texto.

ksmbfs: Cliente de LanManager (protocolo de LAN de Microsoft usado por Windows 3.11, 95 o NT). También se incluye un servidor llamado *samba*.

mc: *Midnight Commander*. Herramienta estilo "Commandante Norton".

unarj, unzip, lha: Descompresores de populares formatos de DOS.

tcsh, bash, zsh, ksh, csh, ash: *Shells* (interpretes de ordenes) más o menos habituales en el mundo Unix.

dosemu: Emulador de DOS. Permite ejecutar sesiones de DOS y muchas de las aplicaciones para él.

pcb: Herramienta para diseño de placas de circuito impreso.

TCL/TK

Para Windows 95, NT y UNIX. Importante para aquellos lectores que

vayan a seguir el curso. La versión Windows 95 se instala ejecutando *win76.exe*.

ADA

Copiar los ficheros zip Gnat3051, Gnat3052, Gnat3053 y Gnat3054, que cuelgan del subdirectorio \ADA\EZ2LOAD, y proceder a descomprimirlos. Será necesario incluir variables de entorno y modificar el path. En el próximo número detallaremos más la instalación, ya que será necesario compilar algún ejemplo de los artículos. De cualquier manera, existe información en la misma distribución.

GUÍA-SP DE INFORMÁTICOS

Nuestro habitual espacio dedicado a mostrar a empresas españolas los currículum de muchos de nuestros lectores-programadores. Desde cualquier navegador se debe cargar *index.htm*. Se anima a todos los lectores a enviar su currículum. Más información en la propia Guía.

DISK29

Los habituales fuentes correspondientes a los artículos de este número 29 de Sólo Programadores.

El programa *dselect* permite seleccionar los paquetes que se quieren instalar en el ordenador, y lo que es más importante, gestiona las dependencias previamente a la instalación. Así, no deja instalar un paquete sin instalar otros que necesite, ni deseleccionar un paquete que otros paquetes seleccionados necesiten.

Este mecanismo de gestión asegura que la instalación es coherente y que no va a fallar posteriormente un programa porque falte una librería. También se comprueba que las versiones de los paquetes sean al menos las mínimas necesarias. Si la instalación no fuese desde cero, sino una actualización de otra versión anterior de Debian, automáticamente se seleccionarían las nuevas versiones de los programas instalados, y tras instalarlos el sistema quedaría exatamente igual que si el antiguo nunca hubiese estado instalado.

Tras culminar la selección de paquetes, el programa comienza a desempaquetar los paquetes elegidos en el disco duro, proceso que durará algunos minutos. Más tarde procederá a configurar los paquetes uno por uno. La mayoría de los paquetes pueden configurarse de manera automática, adaptándose los programas que contienen a la instalación específica en la máquina concreta, pero algunos de ellos realizarán preguntas. Valgan como ejemplos los servidores de World Wide Web, de correo electrónico y News, o el sistema de ventanas XWindow que se muestra en la figura 3.

Este sistema de ventanas, por ejemplo, realizará preguntas específicas acerca del hardware de vídeo que se disponga para aprovechar al máximo las posiblidades que brinde (otro método, comúnmente seguido por otros sistemas de ventanas como Microsoft Windows, prefiere utilizar modos más genéricos de utilización del hardware por facilidad de instalación, a costa de no conseguir aprovecharlo al máximo).

Debe decirse que se puede entrar al programa *dselect* en cualquier otro momento, desde la cuenta de *root*, para instalar nuevos programas, elminar paquetes que se quieran desinstalar, etc...

También puede, en cualquier momento, instalarse un paquete individualmente y con la comprobación de coherencia, a través del uso del programa *dpkg*. Por ejemplo, podríamos instalar el paquete *binutils_2.6-2.deb*, que se encuentra en el directorio *stable/binary-i386/devel* de la distribución, con la orden:

dpkg —install binutils_2.6-2.deb

Una vez terminada la instalación, es necesario adaptar el programa de arranque antes mencionado, LILO, a nuestras necesidades. Para ello hay que proporcionar la descripción de dónde están situados los diferentes sistemas operativos. Basta con editar el archivo */etc/lilo.conf*, por ejemplo con el sencillo editor *ae*:

ae /etc/lilo.conf

y añadir, si por ejemplo se tiene Ms-DOS en la primera partición del primer disco duro:

```
other=/dev/hda1
  table=/dev/hda
  label=DOS
```

resultando finalmente /etc/lilo.conf algo como:

```
boot=/dev/hda
install=/boot/boot.b
map=/boot/map
vga=normal
delay=20
other=/dev/hda1
  table=/dev/hda
  label=DOS
image=/vmlinuz
  label=Linux
  root=/dev/hda2
image=/vmlinuz.old
  label=Linux-old
  root=/dev/hda2
```

Estas modificaciones son necesarias, ya que por defecto la instalación sólo permitiría arrancar Linux. Para hacer los cambios efectivos hay que ejecutar antes de rearrancar:

lilo

En el ejemplo se consigue que al arrancar, tras una breve espera (de 20 decimas de segundo en el ejemplo), LILO inicie automáticamente DOS (el primer sistema de los descritos). Para seleccionar otro, que no sea el arranque por defecto, basta con pulsar <SHIFT> antes que venza el plazo de espera y escribir la etiqueta (*label*) de la opción que se quiera arrancar. Si no se recordasen en ese momento los nombres de las *labels* puede usarse la tecla <TAB> para pedir los nombres válidos y poder elegir uno.

Es importante recordar que, para apagar un máquina que ejecuta Linux, no es conveniente apagar directamente la computadora. Es mejor pulsar la combinación de teclas CTRL+ALT+SUPR (que normalmente sirven para rearrancar), lo que hace que se vacíen los *buffers* y se terminen ordenadamente los procesos antes de apagar.

CORREO DEL LECTOR

En esta sección, los lectores de SÓLO PROGRAMADORES tienen la oportunidad de hallar respuesta a los problemas que puedan tener en cualquier tema relacionado con la programación.

RATÓN DESDE PASCAL

P Me dirijo a vosotros porque llevo muy poco tiempo en esto de la programación y tengo una duda que casi me da vergüenza preguntarla. Estoy haciendo mis primeros pinitos con Turbo Pascal y necesito utilizar el ratón en mi primer programa "serio". Seguro que es muy sencillo, pero no tengo ni la más remota idea de cómo se hace esto. ¿Me podéis explicar cómo se hace? Y de paso, ¿podríais poner algo de código para que me resulte más sencillo?

Mariano Gozalo Arias (Zaragoza)

R Tranquilo, amigo Mariano. Tu pregunta no debe ser motivo de vergüenza bajo ningún concepto. Efectivamente, el ratón es bastante sencillo de utilizar. Existe una interrupción, la 33h, que proporciona una serie de servicios de acceso al ratón. Basta con realizar llamadas a la citada interrupción con los parámetros adecuados para tener al ratón bajo nuestro control. Por desgracia, no disponemos de espacio suficiente en esta sección para explicar todas las llamadas, pero sí podemos incluir una unidad de Turbo Pascal que permite realizar un acceso básico al ratón:

```
Unit Raton;
Interface
Uses dos;

Const RInt=$33; BotonIzq=1; BotonDer=2; BotonCent=4;
CteMov=20;
Procedure MuestraRaton;
Procedure OcultaRaton;
Procedure LeerRaton (var IzqPulsado, DerPulsado, CentPulsado: boolean; var Cx, Cy: word);
Procedure PonerRaton (Cx, Cy: word);
Procedure Pulsado (Boton: word; var Veces, Cx, Cy: word);
Procedure Soltado (Boton: word; var Veces, Cx, Cy: word);
Procedure RangoHor (MinCx, MaxCx: word);
Procedure RangoVert (MinCy, MaxCy: word);
Procedure Movimiento (var hor, vert: integer);

Implementation

Procedure MuestraRaton;
var r: registers;
begin
  r.ax := 1;
  intr (RInt, r);
end;

Procedure OcultaRaton;
var r: registers;
begin
  r.ax := 2;
  intr (RInt, r);
end;

Procedure TransStatus (Reg: word; var IzqPulsado, DerPulsado, CentPulsado: boolean);
begin
IzqPulsado := (Reg and BotonIzq)=BotonIzq;
DerPulsado := (Reg and BotonDer)=BotonDer;
CentPulsado := (Reg and BotonCent)=BotonCent;
end;
Procedure LeerRaton (var IzqPulsado, DerPulsado, CentPulsado: boolean; var Cx, Cy: word);
var r: registers;
begin
  r.ax := 3;
  intr (RInt, r);
    TransStatus (r.bx, IzqPulsado, DerPulsado, CentPulsado);
  Cx := r.cx;
  Cy := r.dx;
end;

Procedure PonerRaton (Cx, Cy: word);
var r: registers;
begin
  r.ax := 4;
  r.cx := Cx;
  r.dx := Cy;
  intr (RInt, r);
end;

Procedure Pulsado (Boton: word; var Veces, Cx, Cy: word);
var r: registers;
begin
  r.ax := 5;
  r.bx := Boton;
  intr (RInt, r);
  Veces := r.bx;
  Cx := r.cx; { Coordenadas en las que se produjo la última pulsación }
  Cy := r.dx;
end;

Procedure Soltado (Boton: word; var Veces, Cx, Cy: word);
```

```
var r: registers;
begin
  r.ax := 6;
  r.bx := Boton;
  intr (RInt, r);
  Veces := r.bx;
  Cx := r.cx; { Coord. en las se dejó de pulsar }
  Cy := r.dx; { el botón por última vez }
end;

Procedure RangoHor (MinCx, MaxCx: word);
var r: registers;
begin
  r.ax := 7;
  r.cx := MinCx;
  r.dx := MaxCx;
  intr (RInt, r);
end;

Procedure RangoVert (MinCy, MaxCy: word);
var r: registers;
begin
  r.ax := 8;
  r.cx := MinCy;
  r.dx := MaxCy;
  intr (RInt, r);
end;

Procedure Movimiento (var hor, vert: integer);
var r: registers;
begin
  r.ax := 11;
  intr (RInt, r);
  hor := r.cx;
  vert := r.dx;
end;

end.
```

Posiblemente con esta unidad sea suficiente pero, en caso de necesitar más información, no es difícil encontrar un libro de programación bajo DOS que dedique unas páginas a la interrupción 33h.

CONSTRUCTORES VIRTUALES EN C++

***P** Soy un asiduo lector de su revista y acudo a ustedes con la sana intención de satisfacer mi curiosidad. Mi pregunta es muy breve: ¿por qué en C++ no se pueden utilizar constructores virtuales?*

Alberto Azorín
(Madrid)

R Su pregunta, aunque breve, no resulta nada fácil de contestar. La razón por la que C++ no admite constructores virtuales habría que preguntársela al mismísimo Bjarne Stroustroup, el diseñador del lenguaje. Posiblemente se debe a que C++ es un superconjunto del lenguaje C, no un lenguaje orientado a objetos "puro" como, por ejemplo, Smalltalk. Lo cierto es que los constructores virtuales no son estrictamente necesarios como sí lo son los destructores virtuales. En cualquier caso, es posible simular el comportamiento de un constructor de este tipo mediante los mecanismos que proporciona el propio C++.

SPRITES

***P** Hola, amigos. Tengo la intención de hacer una pequeña presentación para una tienda en la que trabajo y de paso practico un poco la programación. He empezado a hacer unas pruebas y no sé cómo pintar gráficos en movimiento sobre un fondo sin que éste se borre.*

José Marín Ramos
(Vigo)

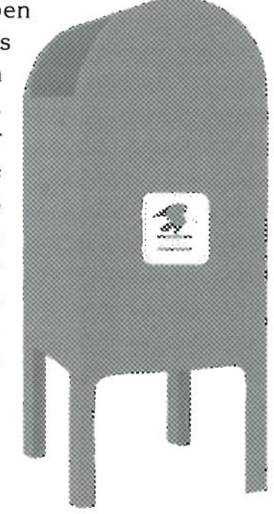
R Lo que usted intenta utilizar son sprites, viejos conocidos de los programadores de juegos. Un sprite es un gráfico, normalmente animado, que pasa por encima de un decorado sin alterarlo. La teoría de la técnica es muy sencilla, pero la práctica no lo es tanto.

Cuando se pinta un gráfico sin utilizar ninguna técnica se observa un cuadrado alrededor del gráfico. Si además este gráfico se mueve, el resultado es un rastro que elimina por completo todo lo que había por debajo. La forma de evitarlo es haciendo transparente las partes del gráfico que sobran. Por ejemplo, en un gráfico de una lupa sobra todo lo que está fuera de la silueta de la misma y lo que corresponde al cristal, que obviamente debe ser transparente. Para lograrlo, se pinta todo lo transparente con un mismo color de la paleta que sólo se utilizará para esos menesteres. A la hora de pintar el gráfico todo lo que tenga ese color simplemente se ignora.

Pero esto no es suficiente, hay que eliminar los rastros dejados por los gráficos. Antes de pintar cada gráfico se debe almacenar en memoria la porción del decorado sobre la que se van a pintar para poder recuperarla más adelante. El proceso sería el siguiente:

1.- Pintar el decorado.
2.- Mientras haya gráficos que pintar:
2.a.- Guardar la porción que el gráfico va a tapar.
2.b.- Pintar el gráfico teniendo en cuenta las partes transparentes.
3.- Recuperar las porciones.
4.- Volver al paso 2.

También aparecerán problemas de parpadeos en la pantalla. El mejor modo de evitarlos es utilizando una memoria intermedia de trabajo. En esa memoria se pinta todo lo que debe aparecer en la pantalla y posteriormente se vuelca a la memoria de vídeo del ordenador. El volcado tiene que hacerse aprovechando el sincronismo vertical del monitor (el momento en el que el haz de luz del monitor acaba de pintar la pantalla por la esquina inferior derecha y retorna a la esquina superior izquierda). Por otro lado, se debe tener en cuenta el orden en que deben aparecer los gráficos en la escena. En primer lugar se vuelcan los gráficos más alejados y a continuación los más cercanos.



CD-ROMs *muy especiales*

Linux Red Hat Commercial v4.0

¡2 CD-ROMs + libro 230 pgs.!

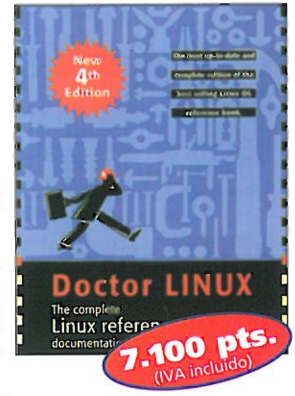
- El Linux más solicitado en la actualidad.
- Versión oficial 4.0 (i386) de Red Hat Software, Inc.
- Incluye licencia para el servidor X de Metro.
- De fácil instalación y mantenimiento.
- Actualizaciones automáticas vía Internet y CD.



Doctor Linux 4ª edición

¡Libro de 1.600 páginas!

- El complemento ideal para el Linux Red Hat.
- Guías y libros de instalación, configuración, mantenimiento y administración.
- De fácil consulta, cuidada selección y organización.



Linux Bible 4ª edición

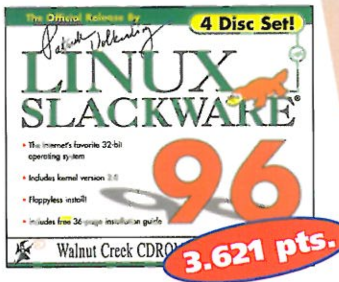
¡CD-ROM + libro 1.800+ pgs.!

- La recopilación de documentación para Linux más completa existente.
- CD-ROM incluyendo el libro completo para búsquedas y consultas.
- Guías y libros de instalación, configuración, mantenimiento y administración.

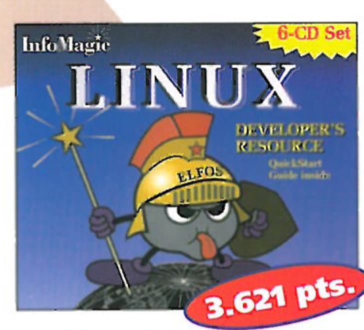
8.500 pts. (IVA incluido)

Linux Slackware 96

¡Edición de agosto/96! ¡Ahora kernel 2.0! ¡4 CD-ROMs!



- Guía impresa de instalación rápida, 36 páginas.
- Recopilación oficial por Patrick Volkerding.



Linux Developer's Resource

¡Edición de septiembre/96! ¡Ahora kernel 2.0! ¡6 CD-ROMs!

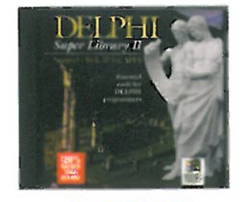
¡¡NOVEDAD!! para Linux

Cyclom-8Zo

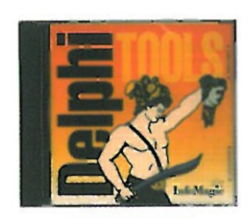
8 puertos serie • Procesador RISC de 32-bits
Bus PCI • 460 Kbps full-duplex por canal

$449

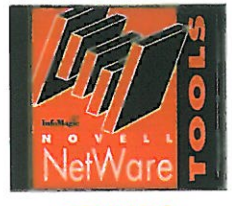
(precio de promoción para distribuidores y proveedores de Internet)



CD0923
Herramientas para programadores Delphi.
2.155 pts.



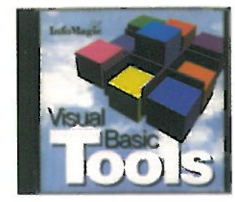
CD0929
Ejemplos, código y documentación para Delphi.
2.413 pts.



CD0958
Herramientas para administradores Novell.
3.017 pts.



CD0892
Herramientas de programación Perl.
3.621 pts.



CD0930
Herramientas de programación Visual Basic.
2.413 pts.

Sistemas Operativos/Programación: ***Disponemos del más amplio y especializado catálogo de CD-ROMs de utilidad para programadores y profesionales. Consúlte nuestro Web.***



abc analog, s.l.
☎ (91) 634 20 00
(91) 634 32 13
FAX (91) 634 47 86
Internet:
http://www.abcnet.es







IVA adicional a los precios. Todas las marcas están registradas por sus respectivos propietarios.